WEIJIYUANLI

JIEKOUYUWANGLUOSHIYONGJISHU

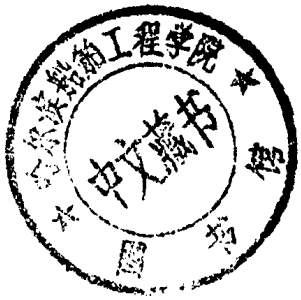
微机原理、接口与网络实用技术

杨绍国　唐　芬　吴宗祥　唐　斌　编著

电子科技大学出版社

微机原理、接口与网络实用技术

杨绍国　唐　芬　吴宗祥　唐　斌　编著



电子科技大学出版社

内容简介

本书以PC系列微机为研究对象，系统地介绍了微机应用系统开发的软硬件接口、微机通信与组网等方面的实用技术。本书的特点是软件与硬件相结合，方法技术与应用实例相结合，以尽可能短的篇幅介绍尽可能多的实用技术，使广大读者在极短的时间内就能掌握微机应用系统设计、微机联网中的关键技术。本书附有大量应用实例，均通过了实践检验，读者可以按本书提供的思路自己设计微机应用系统，自己组建微机网络与通信系统。同时，本书提供了大量的实验，为读者提供了良好的实验环境。

全书共分六章。第一章系统地讨论了微机应用系统中汇编语言程序设计，程序调试及混合编程等实用技术；第二章讨论了微机系统软硬件接口技术；第三章详细介绍了用户接口扩展卡软硬件设计中的关键技术；第四章讨论了微机应用系统设计中的常用接口技术；第五章讨论了微机通信系统中的接口技术；第六章介绍怎样组建局域网。

本书内容充实，构思严谨，并附有大量的应用实例和实验，便于读者学习和借鉴。本书具有广泛的实用性，对于从事微型计算机应用系统开发、微机联网的广大工程技术人员来说，是一本很好的参考书；同时也可作为大专院校《微机原理》、《接口技术》、《计算机控制》、《计算机网络》、《汇编语言程序设计》等课程的教科书或参考书，是一本很好的学习指导书和实验教程。

微机原理、接口与网络实用技术

杨绍国　唐　芬　吴宗祥　唐　斌　编著

*

电子科技大学出版社出版

（成都建设北路二段四号）邮编 610054

电子科技大学出版社印刷厂印刷

新华书店经销

*

开本 787×1092　1/16　印张 14.5　字数 351.4 千字

版次　1996 年 9 月第一版　印次　1996 年 9 月第一次印刷

印数 1—4000 册

ISBN 7—81043—502—7/TP・191

定价：15.00 元

前　言

PC 系列微机是我国目前使用最为广泛的机种，已经渗透到社会各个领域并开始进入家庭。由于这种微机配置了性能优良的外围设备，如显示器、键盘、磁盘、异步串行通信口、打印机等，故可直接用于管理、科学计算、辅助设计等。但由于数据采集与处理、工业控制、网络通信以及多媒体系统等应用领域所需的各类设备接口，微机本身没有也无法提供，这就只能根据应用需要，开发出相应的微机应用系统。

汇编语言是和计算机硬件结合最密切的语言，是微机应用系统软件开发的主要语言，在第一章中详细讨论了汇编语言程序设计、程序调试和混合编程等实用技术。

PC 系列微机一般都配置了显示器、键盘、打印机、磁盘、声音、定时/计数、中断等接口。怎样根据需要来控制它们实现系统软硬件接口？关于这个问题将在第二章讨论。

虽然计算机系统为用户提供了一些常用的接口，但是微机应用领域广泛，用途各异，常常需要用户自行设计和开发满足特定需求的接口控制卡。第三章和第四章专门讨论微机用户接口扩展卡的软硬件设计问题。

随着微型计算机在各行各业的普遍应用，微机与微机之间以及微机与应用系统之间的信息交换便成为一个十分迫切而实际的问题，解决这个问题的方法在第五章中讨论。

80 年代以来，PC 技术及产品的迅速发展，推动了 PC 组网技术的飞速发展。目前，PC 组网已进入到各种应用领域。怎样组建局域网以便实现软硬件资源共享是一个需要解决的实际问题。本书的第六章专门介绍了局域网的组网技术，书中所列方法均通过了实践检验，读者可以直接移植。

本书在编写过程中，参考了许多书刊和文献资料，在此向提供帮助的各位作者表示感谢。

由于编者水平有限，错误及不当之处，在所难免，敬请广大读者批评指正。

编　者

一九九五年十二月于成都

目　录

第一章　汇编语言程序设计

第二章　系统软硬件接口技术

第三章 用户接口扩展软硬件设计

第四章 微机应用系统常用接口

第五章 微机通信接口技术

第六章 自己动手组建局域网

第一章　汇编语言程序设计

1.1　汇编语言程序的上机操作

1.1.1　汇编语言程序上机过程

汇编语言的上机过程如图 1.1 所示，现以后面讲到的例 1.1 为例分框说明如下：

一、用编辑程序建立和修改源程序(ASM 文件)

汇编源程序是文本文件，可以用任何文本文件的编辑软件来建立和修改汇编源程序。常用的有 EDLIN、WORDSTAR、WPS(N 命令)、NE 等编辑软件。例如用 EDLIN 编辑：

C>EDLIN EXAM11. ASM↙

二、用 ASM 或者 MASM 命令产生目标文件(OBJ 文件)

例如，对 EXAM11. ASM 汇编

C>MASM↙

此时，汇编程序给出如下回答：

```
Microsoft (R) Macro Assembler Version 5.00
Copyright (C) Microsoft Corp 1981-1985,1987. All rights reserved.
Source filename [.ASM]:EXAM11
Object filename [EXAM11.OBJ]:
Source listing [NUL.LST]:EXAM11
Cross-reference[NUL.CRF]:EXAM11
50628+272508 Bytes symbol space free
  0 Warning Errors
  0 Severe Errors
```

从上面的操作过程中可以见到，汇编程序的输入文件就是用户编写的汇编语言源程序，它必须以 ASM 为文件扩展名。汇编程序的输出文件有三个，第一个是目标文件，它以 OBJ 为扩展名，产生 OBJ 文件是我们进行汇编操作的主要目的，所以这个文件是一定要产生，也一定会产生的，操作时，这一步只要打入回车就行了；第二个是列表文件，它以 LST 为扩展名，列表文件同时给出源程序和机器语言程序，从而，可以使调试变得方便，列表文件是可有可无的，如果不需要，则在屏幕上出现提示信息[NUL. LST]:时，打入回车即可，如果需要，则打入文件名和回车；第三个是交叉符号表，此表给出了用户定义的所有符号，对每个符号都列出了将其定义的所在行号和引用的行号，并在定义行号上加上"#"号，同列表文件一样，交叉符号表也是为了便于调试而设置的，对于一些规模较大的程序，交叉符号表为调试工作带来很大方便，当然，交叉符号表也是可有可无的，如不需要，那么，在屏幕上出现提示信息

[NUL. CRF]:时，打入回车即可。

汇编过程结束时，会给出源程序中的警告性错误[Warning Errors]和严重错误[Severs Errors]，前者指出一般性错误，后者指出语法性错误，当存在这两类错误时，屏幕上除指出错误个数外，还给出错误信息代号，程序员可以通过查找手册弄清错误的性质。

如果汇编过程中，发现有错误，则程序员应该重新用编辑命令修改错误，再进行汇编，最终直到汇编正确通过。要指出的是汇编过程只能指出源程序中的语法错误，并不能指出算法错误和其他错误。

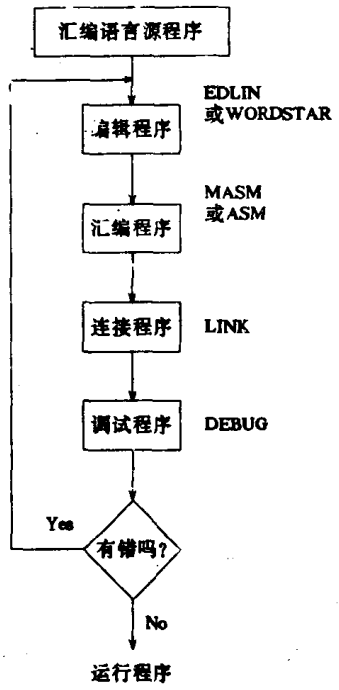


图 1.1 汇编语言过程流程图

三、用 LINK 命令产生执行文件(EXE 文件)

由汇编程序建立的目标码文件必须经过连接后，才能成为可执行文件。连接过程如下：

C>LINK ↙

此时屏幕上见到如下回答信息：

```
Microsoft (R) Overlay Linker Version 3.60
Copyright (C) Microsoft Corp 1983-1987. All rights reserved.
Object Modules [.OBJ]:EXAM11
Run File [EXAM11.EXE]:
List File [NUL.MAP]:EXAM11
Libraries [.LIB]:
```

LINK 命令有一个输入文件，即 OBJ 文件，有时，用户程序用到库函数，此时，对于提示信息 Libraries[. LIB]，要输入库名。

LINK 过程产生两个输出文件，一个是扩展名为 EXE 的执行文件，产生此文件当然是 LINK 过程的主要目的；另一个是扩展名为 MAP 的列表分配文件，有人也称它为映像文件，它给出每个段在内存中的分配情况。MAP 文件是可有可无的。

四、用 DEBUG 调试程序

由 LINK 命令产生的可执行文件在 DOS 操作系统下就可以运行了。但是，绝大多数程序初次运行结果都是不对的，也就是说存在运算错误，运算错误相对于 MASM、LINK 程序发现的语法错误要难发现得多，它必须通过 DEBUG 程序来进行调试，才能发现和改正错误。

1.1.2 程序段前缀 PSP

汇编语言程序经过汇编、连接以后生成的 EXE 文件，可在 PC-DOS 支持下装入内存，并从程序中指定的地址开始运行。装入文件并设置启动地址由 PC-DOS 的 COMMAND. COM 文件的 EXEC 来完成的。COMMAND. COM 在装入 EXE 文件前，首先确定最低内存可用地址

作为被装入程序的可用内存起点;再在这个程序段内偏移地址 0000H 处构造一个 0100H 字节的程序段前缀 PSP(Program Segment Prefix)控制块。EXEC 在这个程序段的偏移量 0100H 处开始装载被调用的程序,并把控制交给 0100H 处的指令。

程序段前缀 PSP 实际上是一个程序控制块,PC-DOS 建立这个程序控制块,并且利用它来管理系统的进程。当一个程序获得控制时,PSP 的结构如表 1-1:

表 1-1 PSP 的结构

字段范围 (16 进制)	字段长度 (10 进制)	字段含义
0—1 字节	2	指令 INT20H
2—3 字节	2	可用内存空间(以 16 字节为单位)
4 字节	1	保留
5—9 字节	5	远调用指令(Call 功能调用入口)
A—D 字节	4	程序结束地址(INT22H 入口)
E—11 字节	4	Ctrl-Break 出口地址(INT23H 入口)
12—15 字节	4	标准错误出口地址(INT24H 入口)
16—2B 字节	22	保留
2C—2D 字节	2	传送环境的段地址
2E—31 字节	4	保留
32—4F 字节	30	保留给 DOS 专用
50—51 字节	2	DOS 功能调用 INT21H 指令
52 字节	1	远返回指令
53—5B 字节	9	保留
5C—6B 字节	16	格式化参数 1,为未打开的 FCB_1 使用
6C—7B 字节	16	格式化参数 2,为未打开的 FCB_2 使用
7C—7F	4	保留
80 字节	1	约定的磁盘传送地址区或非格式化参数长度
81—FF 字节	127	约定的磁盘传送地址区或格式化参数

0—1 字节存放一条 DOS 程序结束的指令,当被调入的用户程序段执行完后返回到此字段,则由 INT20H 指令将控制权送回给 DOS,这是用户程序回到 DOS 的一种方法。

2—3 字节中存放当前可用内存的总空间,以 16 字节为单位,例如 1000H 表示 64K 字节。

5—9 字节放一条长调用指令 Call。在 5 字节处存放 Call 指令码 9AH,6—9 字节存放 DOS 系统功能调用的入口(即 INT21H 的入口)。因此,在用户程序中只需发一个近调用到 PSP 的位移 5 处,并把调用的子程序号送 AH,即可进入该调用的子程序。如果 AH=00 或 4CH,则进入"程序结束"子程序,而把控制权返回给 DOS,这是从用户程序回到 DOS 的又一种方法。

A—D,E—11,12—15等三个字段是进入用户程序段时,DOS将相应的三个中断入口地址存放在此(低字节放位移,高字节放段值),以便在用户程序中可以建立用户自己的结束程序地址、Ctrl-Break的出口以及出错处理程序。当然,这需要在用户程序中修改相应中断向量表,但不管怎样,当用户程序运行结束时,上述三个向量相应地用PSP在此三个字段所保存的值来加以恢复。

2C—2D字节包含传送环境块段地址,环境块包含许多ASCII字符串(许多以空格和零字节结尾的ASCII串)。DOS规定环境的全长要小于32K,其格式为"名字=参数"的ASCII字符串组成,且规定每个字符串的最后一个字节为00H;整个字串集也用00H结尾。环境由命令处理程序建立的。所以引入环境这个概念,也是为使用上的方便,比如在程序运行过程中,用户想随时能加入一些信息,就可以使用这种方式,通过有关DOS命令(比如SET)建立一个环境。

5C—6B,6C—7B字段存放两个未打开的文件控制块FCB_1,FCB_2(FCB的格式和功能下面将介绍),它们被解释为DOS外部命令或可执行的程序名之后的参数部分,其一般形式为:

文件名　参数1　参数2

其中,文件名的扩展名必须是.COM或.EXE。这样,当把参数1,2假定为文件名时,则PSP前缀块按FCB格式将两个文件名分别格式化成未打开的FCB,用户便可在调入的用户程序段内对FCB进行所需的操作。由于早期为和CP/M兼容,这种缺省的FCB很有用,但现在DOS必须给应用程序提供完善的路径支持,FCB型文件不支持多级目录结构,因而缺省的FCB便失去了它的作用。

80—FF字段有两个作用:一是用作系统约定的磁盘传送地址区DTA,另一是存放跟在上述文件名之后的全部参数字符,称作非格式化参数区。

所谓DTA,就是磁盘传输时系统约定的缓冲区,起始地址在80H处,其容量为128个字节。用户也可以通过功能调用1AH,在内存任何位置设定自己的DTA。在用户程序中凡使用磁盘I/O的子程序时,都应先设置好DTA,且一次只能有一个DTA是有效的。在用1AH子程序定义为另一个有效的DTA前,所有的磁盘操作都使用这个有效的DTA。

以上说明使我们对PSP的各段含义和作用有一个大致了解,但涉及到有关寄存器的值还没给出。这些寄存器的值随着本段程序的属性不同,而有不同的解释:

对使用扩展名.EXE的程序(图1.2)

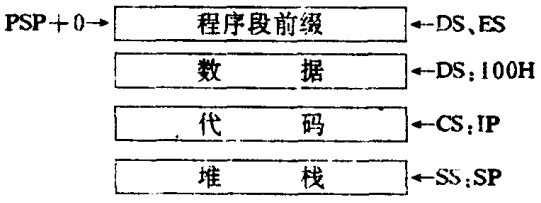


图1.2　EXE文件的执行

(1)DS、ES寄存器指向程序段前缀(用PSP+0表示)而不是指向用户程序的数据段和附加段,而数据段起始地址为DS:100H。

(2)CS、IP、SS和SP寄存器的值是由连接程序(Link)传递过来的。

对使用扩展名.COM的程序(图1.3)

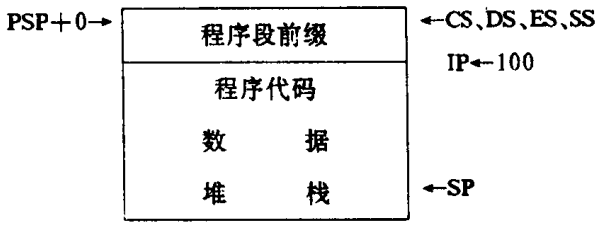


图 1.3　COM 文件的执行

(1)所有四个段寄存器都包含有程序前缀段的段地址,即 DOS 分配的程序段首地址(用 PSP+0 表示)这时内存全部用户区都分配给用户程序。

(2)指令指针 IP 被置为 100H,也就是说从程序段的第一条指令开始执行。

(3)在 PSP 的位移 6 处包含有程序段的可用字节数。

(4)SP 寄存器被置到程序段的末尾。

1.1.3　怎样用 DEBUG 调试程序

用 DEBUG 程序装入.EXE 的程序后,在程序尚未运行之前,各段寄存器的初值为:DS、ES 寄存器指向程序段前缀,因此,DS:100H 和 ES:100H 指向数据段。CS、SS 分别指向代码段和堆栈段。要想 DS 和 ES 分别指向用户数据段和附加段,用户程序必须重新设置,而 CS 和 SS 则不必。下面我们以例 1.1 的程序为例说明怎样用 DEBUG 调试程序。

一、怎样查看和修改寄存器的内容

为了了解程序运行的正确性,常常需要检查寄存器的内容,R 命令就是用来检查寄存器的内容,它有如下三种功能:

1. 显示和修改一个指定寄存器的内容,其格式为:

　　R〈寄存器〉

2. 显示所有寄存器的内容和全部标志位的状态,其格式为:

　　R

3. 显示和修改所有标志位的状态,其格式为:

　　RF

例如,

```
D>DEBUG EXAM11.EXE
-R
AX=0000  BX=0000  CX=002E  DX=0000  SP=0014  BP=0000  SI=0000  DI=0000
DS=13F2  ES=13F2  SS=1405  CS=1403  IP=0000   NV UP EI PL NZ NA PO NC
1403:0000  1E                PUSH    DS
-R  AX
AX  0000
:FFFF
-R
AX=FFFF  BX=0000  CX=002E  DX=0000  SP=0014  BP=0000  SI=0000  DI=0000
DS=13F2  ES=13F2  SS=1405  CS=1403  IP=0000   NV UP EI PL NZ NA PO NC
```

```
1403:0000  1E                PUSH    DS
-RF
NV UP EI PL NZ NA PO NC -OV DI CY
-RF
OV UP DI PL NZ NA PO CY -
-Q
```

在上例中显示了如下操作:(1)用 R 命令查看全部寄存器的内容和 CS:IP 所指的指令(PUSH DS)、指令机器码(1E)和所在存储单元的段地址和偏移地址(1403:0000);(2)用 R 命令把 AX 寄存器的内容由 0000H 改为 FFFFH,并查看改正后 AX 的值;(3)用 RF 命令修改了标志寄存器的值(NV 改为 OV,EI 改为 DI,NC 改为 CY)并查看修改后的结果。

二、怎样查看和修改代码段的内容

1. 怎样查看代码段的内容

在程序调试过程中,经常需要查看内存中装入的程序,这可用反汇编命令 U 完成,其格式为:

U〈起始地址〉L〈长度〉

若省略起始地址,则起始地址为上一次 U 命令反汇编过的最后一条指令后面的地址。若前面没有输入过 U 命令,则起始地址为 CS:IP。若段地址为 CS,则可以省略。若省略地址长度,则为 32 个字节。例如,

```
D>DEBUG EXAM11.EXE
-P=0  2
AX=0000  BX=0000  CX=002E  DX=0000  SP=0012  BP=0000  SI=0000  DI=0000
DS=13F2  ES=13F2  SS=1405  CS=1403  IP=0001   NV UP EI PL NZ NA PO NC
1403:0001  B80000            MOV     AX,0000

AX=0000  BX=0000  CX=002E  DX=0000  SP=0012  BP=0000  SI=0000  DI=0000
DS=13F2  ES=13F2  SS=1405  CS=1403  IP=0004   NV UP EI PL NZ NA PO NC
1403:0004  50                PUSH    AX
-U
1403:0004  50                PUSH    AX
1403:0005  B80214            MOV     AX,1402
1403:0008  8ED8              MOV     DS,AX
1403:000A  BB0000            MOV     BX,0000
1403:000D  8A07              MOV     AL,[BX]
1403:000F  43                INC     BX
1403:0010  0207              ADD     AL,[BX]
1403:0012  43                INC     BX
1403:0013  8A0F              MOV     CL,[BX]
1403:0015  43                INC     BX
1403:0016  020F              ADD     CL,[BX]
1403:0018  2AC1              SUB     AL,CL
1403:001A  43                INC     BX
```

```
1403:001B  8807          MOV     [BX],AL
1403:001D  CB            RETF
1403:001E  41            INC     CX
1403:001F  63            DB      63
1403:0020  7469          JZ      008B
1403:0022  7665          JBE     0089
-U  0008 L 4
1403:0008  8ED8          MOV     DS,AX
1403:000A  BB0000        MOV     BX,0000
-U
1403:000D  8A07          MOV     AL,[BX]
1403:000F  43            INC     BX
1403:0010  0207          ADD     AL,[BX]
1403:0012  43            INC     BX
1403:0013  8A0F          MOV     CL,[BX]
1403:0015  43            INC     BX
1403:0016  020F          ADD     CL,[BX]
1403:0018  2AC1          SUB     AL,CL
1403:001A  43            INC     BX
1403:001B  8807          MOV     [BX],AL
1403:001D  CB            RETF
1403:001E  41            INC     CX
1403:001F  63            DB      63
1403:0020  7469          JZ      008B
1403:0022  7665          JBE     0089
1403:0024  20636F        AND     [BP+DI+6F],AH
1403:0027  64            DB      64
1403:0028  0000          ADD     [BX+SI],AL
1403:002A  0000          ADD     [BX+SI],AL
1403:002C  0400          ADD     AL,00
```

关于上例，作如下几点说明：(1)指令 RETF 之前的内容才是本程序的指令，RETF 之后的内容是内存中其他程序的指令；(2)由 U 命令只能看到指令性语句，对于指示性语句，是不形成机器代码的，因此用 U 命令是看不到的；(3)源程序中的标号和名字，在此均换成了它们所代表的地址。

2. 怎样修改代码段的内容

在程序调试过程中，若发现某些指令有错，这时需要修改指令，可用汇编命令 A 来完成，其格式如下：

A〈地址〉

若命令中没有指定地址，则为上一个 A 命令的最后一个单元之后的地址；若前面没有用过 A 命令，则为 CS:0000H。若段地址为 CS，可以省略。例如，要把指令 SUB AL,CL 改为 ADD AL,CL，该指令存放的开始地址为 DS:0018H，则可按如下步骤完成：

```
-A  0018
```

```
1403:0018  ADD  AL,CL
1403:001A
-U0
1403:0000  1E              PUSH   DS
1403:0001  B80000          MOV    AX,0000
1403:0004  50              PUSH   AX
1403:0005  B80214          MOV    AX,1402
1403:0008  8ED8            MOV    DS,AX
1403:000A  BB0000          MOV    BX,0000
1403:000D  8A07            MOV    AL,[BX]
1403:000F  43              INC    BX
1403:0010  0207            ADD    AL,[BX]
1403:0012  43              INC    BX
1403:0013  8A0F            MOV    CL,[BX]
1403:0015  43              INC    BX
1403:0016  020F            ADD    CL,[BX]
1403:0018  00C8            ADD    AL,CL
1403:001A  43              INC    BX
1403:001B  8807            MOV    [BX],AL
1403:001D  CB              RETF
1403:001E  41              INC    CX
1403:001F  63              DB     63
```

三、怎样查看和修改数据段的内容

程序所需的数据和最终运算结果通常是保存在数据段内，故常需要查看和修改数据段的内容。

1. 怎样查看数据段的内容

查看数据段的内容，可用 D 命令来完成，其格式如下

D〈起始地址〉L〈长度〉

若起始地址省略，则为上一个 D 命令显示最后一个单元后面的地址，如果以前没有用过 D 命令，则约定地址为 DS:100H。D 命令约定段地址为 DS。

若长度省略，则显 80H 个字节。

在 DEBUG 装入 .EXE 程序之后，在数据段寄存器重新定向之前，DS:100H 指向数据段，故查看数据段的内容可用如下命令：

D DS:100 L〈长度〉

或 D 100 L〈长度〉

例如：

```
D>DEBUG EXAM11.EXE
-D DS:100 L 20
13F2:0100  13 27 11 12 00 00 00 00-00 00 00 00 00 00 00 00   .'..............
13F2:0110  1E B8 00 00 50 B8 02 14-8E D8 BB 00 00 8A 07 43   ....P..........C
```

```
-D 100 L 20
13F2:0100  13 27 11 12 00 00 00 00-00 00 00 00 00 00 00 00    .'..............
13F2:0110  1E B8 00 00 50 B8 02 14-8E D8 BB 00 00 8A 07 43    ....P..........C
```

若用 G 命令运行程序，则仍可用上述方法来查看存放在数据段的运行结果。例如：

```
D>DEBUG EXAM11.EXE
-G=0

Program terminated normally
-D DS:100 L 20
13F2:0100  13 27 11 12 17 00 00 00-00 00 00 00 00 00 00 00    .'..............
13F2:0110  1E B8 00 00 50 B8 02 14-8E D8 BB 00 00 8A 07 43    ....P..........C
```

从结果单元可见，其值为 17H，运行结果正确。

在程序运行之后，若 DS 重新定义，指向用户数据段，则查看数据段内容可用如下命令：

D DS:0000 L〈长度〉

或 D 0 L〈长度〉

上述命令也用于查看存放在数据段内由单步运行方式所得到的运行结果。例如：

```
-T=0 5

AX=0000  BX=0000  CX=002E  DX=0000  SP=0012  BP=0000  SI=0000  DI=0000
DS=13F2  ES=13F2  SS=1405  CS=1403  IP=0001   NV UP EI PL NZ NA PO NC
1403:0001  B80000          MOV     AX,0000

AX=0000  BX=0000  CX=002E  DX=0000  SP=0012  BP=0000  SI=0000  DI=0000
DS=13F2  ES=13F2  SS=1405  CS=1403  IP=0004   NV UP EI PL NZ NA PO NC
1403:0004  50              PUSH    AX

AX=0000  BX=0000  CX=002E  DX=0000  SP=0010  BP=0000  SI=0000  DI=0000
DS=13F2  ES=13F2  SS=1405  CS=1403  IP=0005   NV UP EI PL NZ NA PO NC
1403:0005  B80214          MOV     AX,1402

AX=1402  BX=0000  CX=002E  DX=0000  SP=0010  BP=0000  SI=0000  DI=0000
DS=13F2  ES=13F2  SS=1405  CS=1403  IP=0008   NV UP EI PL NZ NA PO NC
1403:0008  8ED8            MOV     DX,AX

AX=1402  BX=0000  CX=002F  DX=0000  SP=0010  BP=0000  SI=0000  DI=0000
DS=1402  ES=13F2  SS=1405  CS=1403  IP=000A   NV UP EI PL NZ NA PO NC
1403:000A  BB0000          MOV     BX,0000
-D  0  L  20
1402:0000  13 27 11 12 00 00 00 00-00 00 00 00 00 00 00 00    .'..............
1402:0010  1E B8 00 00 50 B8 02 14-8E D8 BB 00 00 8A 07 43    ....P..........C
-P  A
```

```
AX=1402  BX=0000  CX=002E  DX=0000  SP=0010  BP=0000  SI=0000  DI=0000
DS=1402  ES=13F2  SS=1405  CS=1403  IP=000D  NV UP EI PL NZ NA PO NC
1403:000D  8A07              MOV     AL,[BX]                          DS:0000=13

AX=1413  BX=0000  CX=002E  DX=0000  SP=0010  BP=0000  SI=0000  DI=0000
DS=1402  ES=13F2  SS=1405  CS=1403  IP=000F  NV UP EI PL NZ NA PO NC
1403:000F  43                INC     BX

AX=1413  BX=0001  CX=002E  DX=0000  SP=0010  BP=0000  SI=0000  DI=0000
DS=1402  ES=13F2  SS=1405  CS=1403  IP=0010  NV UP EI PL NZ NA PO NC
1403:0010  0207              ADD     AL,[BX]                          DS:0001=27

AX=143A  BX=0001  CX=002E  DX=0000  SP=0010  BP=0000  SI=0000  DI=0000
DS=1402  ES=13F2  SS=1405  CS=1403  IP=0012  NV UP EI PL NZ NA PO NC
1403:0012  43                INC     BX

AX=143A  BX=0002  CX=002E  DX=0000  SP=0010  BP=0000  SI=0000  DI=0000
DS=1402  ES=13F2  SS=1405  CS=1403  IP=0013  NV UP EI PL NZ NA PO NC
1403:0013  8A0F              MOV     CL,[BX]                          DS:0002=11

AX=143A  BX=0002  CX=0011  DX=0000  SP=0010  BP=0000  SI=0000  DI=0000
DS=1402  ES=13F2  SS=1405  CS=1403  IP=0015  NV UP EI PL NZ NA PO NC
1403:0015  43                INC     BX

AX=143A  BX=0003  CX=0011  DX=0000  SP=0010  BP=0000  SI=0000  DI=0000
DS=1402  ES=13F2  SS=1405  CS=1403  IP=0016  NV UP EI PL NZ NA PO NC
1403:0016  020F              ADD     CL,[BX]                          DS:0003=12

AX=143A  BX=0003  CX=0023  DX=0000  SP=0010  BP=0000  SI=0000  DI=0000
DS=1402  ES=13F2  SS=1405  CS=1403  IP=0018  NV UP EI PL NZ NA PO NC
1403:0018  2AC1              SUB     AL,CL

AX=1417  BX=0003  CX=0023  DX=0000  SP=0010  BP=0000  SI=0000  DI=0000
DS=1402  ES=13F2  SS=1405  CS=1403  IP=001A  NV UP EI PL NZ NA PO NC
1403:001A  43                INC     BX

AX=1417  BX=0004  CX=0023  DX=0000  SP=0010  BP=0000  SI=0000  DI=0000
DS=1402  ES=13F2  SS=1405  CS=1403  IP=001B  NV UP EI PL NZ NA PO NC
1403:001B  8807              MOV     [BX],AL                          DS:0004=00

AX=1417  BX=0004  CX=0023  DX=0000  SP=0010  BP=0000  SI=0000  DI=0000
DS=1402  ES=13F2  SS=1405  CS=1403  IP=001D  NV UP EI PL NZ NA PO NC
1403:001D  CB                RETF
```

```
-D  0  L  20
1402:0000  13 27 11 12 17 00 00 00-00 00  00 00 00 00 00 00     .'..............
1402 0010  1E B8 00 00 50 B8 02 14-8E D8 BB 00 00 8A 07 43     ....P..........C
-P

AX=1417  BX=0004  CX=0023  DX=0000  SP=0014  BP=0000  SI=0000  DI=0000
DS=1402  ES=13F2  SS=1405  CS=13F2  IP=0000   NV UP EI PL NZ NA PO NC
13F2:0000  CD20              INT     20
-P

Program terminated normally
-D  0  L  20
1402:0000  13 27 11 12 17 00 00 00-00 00  00 00 00 00 00 00     .'..............
1402 0010  1E B8 00 00 50 B8 02 14-8E D8 BB 00 00 8A 07 43     ....P..........C
```

2. 怎样修改数据段的内容

修改数据段的内容可用 E 命令、A 命令和 F 命令，A 命令前面已讲过，E 命令和 F 命令格式如下：

E〈地址〉〈内容表〉

F〈地址范围〉〈内容〉

其功能是用命令中指定的内容去代替指定范围的内存单元的内容。内容表是一系列用空格隔开的 16 进制字节，或者用引号括起来的字符串。E 命令中地址范围大小等于内容表中给定内容的字节数。F 命令中，若内容表的长度小于给定地址范围长度，则重复使用内容表的内容。相反，若内容表的长度大于给定地址范围的长度，则略去多余的值。例如：

```
D>DEBUG EXAM11.EXE
-D  100  L  20
13F2:0100  13 27 11 12 00 00 00 00-00 00  00 00 00 00 00 00     .'..............
13F2 0110  1E B8 00 00 50 B8 02 14-8E D8 BB 00 00 8A 07 43     ....P..........C
-E  100  12 26 10 13
-D  100  L  20
13F2:0100  13 26 10 13 00 00 00 00-00 00  00 00 00 00 00 00     .&..............
13F2 0110  1E B8 00 00 50 B8 02 14-8E D8 BB 00 00 8A 07 43     ....P..........C
-F  100  L  4 13 27 11 12
-D  100  L  20
13F2:0100  13 27 11 12 00 00 00 00-00 00  00 00 00 00 00 00     .'..............
13F2 0110  1E B8 00 00 50 B8 02 14-8E D8 BB 00 00 8A 07 43     ....P..........C
-F  100  L  8 'AB',10
-D  100  L  20
13F2:0100  41 42 10 41 42 10 41 42-00 00  00 00 00 00 00 00     AB. AB. AB.....
13F2 0110  1E B8 00 00 50 B8 02 14-8E D8 BB 00 00 8A 07 43     ....P..........C
-F  100  L  4  'ABCDBAB',10
-D  100  L  20
13F2:0100  41 42 43 44 42 10 41 42-00 00  00 00 00 00 00 00     ABCDB. AB.....
```

```
13F2 0110  1E B8 00 00 50 B8 02 14-8E D8 BB 00 00 8A 07 43    ....P.........C
-A  DS:100
13F2:0100  DB  13,27,11,12
13F2:0104
-D  100  L  20
13F2:0100  13 27 11 12 42 10 41 42-00 00  00 00 00 00 00 00    .'..B.AB........
13F2 0110  1E B8 00 00 50 B8 02 14-8E D8 BB 00 00 8A 07 43    ....P.........C
```

四、怎样用 DEBUG 调试程序

DEBUG 为用户提供了二种程序调试手段：

1. 设置断点

为了对程序进行逐段调试，希望在运行中能设断点，以便分段检查程序运行的正确性。G 命令即具有这种功能，其格式如下：

G=〈运行程序起始地址〉〈断点地址〉…〈断点地址〉

若起始地址省略，则为 CS:IP。约定段地址为 CS。现举例如下：

```
D>DEBUG EXAM11.EXE
-G=0  0

AX=0000  BX=0000  CX=002E  DX=0000  SP=0014  BP=0000  SI=0000  DI=0000
DS=13F2  ES=13F2  SS=1405  CS=1403  IP=0000  NV UP EI PL NZ NA PO NC
1403:0000  1E              PUSH    DS
-
D>DEBUG EXAM11.EXE
-G=0  16

AX=143A  BX=0003  CX=0011  DX=0000  SP=0010  BP=0000  SI=0000  DI=0000
DS=1402  ES=13F2  SS=1405  CS=1403  IP=0016  NV UP EI PL NZ NA PO NC
1403:0016  020F            ADD     CL,[BX]                    DS:0003=12
-G=0
D>DEBUG EXAM11.EXE
-G=0  18

AX=143A  BX=0003  CX=0023  DX=0000  SP=0010  BP=0000  SI=0000  DI=0000
DS=1402  ES=13F2  SS=1405  CS=1403  IP=0018  NV UP EI PL NZ NA PO NC
1403:0018  2AC1            SUB     AL,CL
-Q
```

该例中，第一个断点设在 CS:0000H，显示程序的主要部分运行以前各寄存器的情况，以备用户查看。第三个断点设在 CS:0018H，用户可查看寄存器 AL 和 CL 中的运算结果。AL 的值为 3A，CL 的值为 23，结果正确。

2. 设置单步工作方式

利用 T 命令和 P 命令可逐条跟踪程序的执行，并在执行每条指令后，显示各寄存器的内容和标志位的状态，以便检查运行结果的正确性。其命令格式如下：

T=〈起始地址〉〈指令条数〉

P=〈起始地址〉〈指令条数〉

起始地址省略，则为 CS:IP。指令条数省略，则为 1。约定地址为 CS。例如：

```
-T
AX=143A  BX=0003  CX=0023  DX=0000  SP=0010  BP=0000  SI=0000  DI=0000
DS=1402  ES=13F2  SS=1405  CS=1403  IP=0018  NV UP EI PL NZ NA PO NC
1403:0018  2AC1              SUB    AL,CL
-T

AX=1417  BX=0003  CX=0023  DX=0000  SP=0010  BP=0000  SI=0000  DI=0000
DS=1402  ES=13F2  SS=1405  CS=1403  IP=001A  NV UP EI PL NZ NA PO NC
1403:001A  43                INC    BX
-T

AX=1417  BX=0004  CX=0023  DX=0000  SP=0010  BP=0000  SI=0000  DI=0000
DS=1402  ES=13F2  SS=1405  CS=1403  IP=001B  NV UP EI PL NZ NA PO NC
1403:001B  8807              MOV    [BX],AL                    DS:0004=00
-T

AX=1417  BX=0004  CX=0023  DX=0000  SP=0010  BP=0000  SI=0000  DI=0000
DS=1402  ES=13F2  SS=1405  CS=1403  IP=001D  NV UP EI PL NZ NA PO NC
1403:001D  CB                RETF
-T

AX=1417  BX=0004  CX=0023  DX=0000  SP=0014  BP=0000  SI=0000  DI=0000
DS=1402  ES=13F2  SS=1405  CS=13F2  IP=0000  NV UP EI PL NZ NA PO NC
13F2:0000  CD20              INT    20
-T

AX=1417  BX=0004  CX=0023  DX=0000  SP=000E  BP=0000  SI=0000  DI=0000
DS=1402  ES=13F2  SS=1405  CS=0293  IP=143F  NV UP EI PL NZ NA PO NC
0293:143F  B400              MOV    AH,00
-T

AX=0017  BX=0004  CX=0023  DX=0000  SP=000E  BP=0000  SI=0000  DI=0000
DS=1402  ES=13F2  SS=1405  CS=0293  IP=1441  NV UP EI PL NZ NA PO NC
0293:1441  EB24              JMP    1467
-T

-P

AX=143A  BX=0003  CX=0023  DX=0000  SP=0010  BP=0000  SI=0000  DI=0000
DS=1402  ES=13F2  SS=1405  CS=1403  IP=0018  NV UP EI PL NZ NA PO NC
```

```
1403:0018  2AC1              SUB     AL,CL
-P

AX=1417  BX=0003  CX=0023  DX=0000  SP=0010  BP=0000  SI=0000  DI=0000
DS=1402  ES=13F2  SS=1405  CS=1403  IP=001A   NV UP EI PL NZ NA PO NC
1403:001A  43                INC     BX
-P

AX=1417  BX=0004  CX=0023  DX=0000  SP=0010  BP=0000  SI=0000  DI=0000
DS=1402  ES=13F2  SS=1405  CS=1403  IP=001B   NV UP EI PL NZ NA PO NC
1403:001B  8807              MOV     [BX],AL                    DS:0004=00
-P

AX=1417  BX=0004  CX=0023  DX=0000  SP=0010  BP=0000  SI=0000  DI=0000
DS=1402  ES=13F2  SS=1405  CS=1403  IP=001D   NV UP EI PL NZ NA PO NC
1403:001D  CB                RETF
-P

AX=1417  BX=0004  CX=0023  DX=0000  SP=0014  BP=0000  SI=0000  DI=0000
DS=1402  ES=13F2  SS=1405  CS=13F2  IP=0000   NV UP EI PL NZ NA PO NC
13F2:0000  CD20              INT     20
-P

Program terminated normally
```

由该例可见，除指令 INT 20H 以外，对其他指令 P 命令和 T 命令的功能是一样的。实事上，除循环、重复串指令、软中断和子程序调用以外的指令，P 命令和 T 命令的功能是一样的。P 命令把循环、重复串指令、软中断和子程序调用看作一个单位指令予以执行；而 T 命令把它们看作多条指令予以执行，每次只执行其中之一条指令。

1.2 简单程序设计

1.2.1 汇编语言程序设计的基本步骤

一、程序设计的基本要求

1. 要能实现预定的功能，能正常运行；
2. 程序要结构化，可读性好；
3. 运行速度快；
4. 占用存储空间少。

二、汇编程序设计的基本步骤

1. 分析问题

拿到一个题目，首先要全面分析它，由此归纳出解决问题的数学模型。

2. 确定解决问题的算法。

根据数学模型，拟定最佳的计算方法和操作步骤。

3. 绘制流程图

根据算法思想将解决问题的方法以框图的形式表述出来。

4. 分配存储空间和工作单元

根据要求定义数据段、堆栈段、代码段以及附加段。工作单元可以设置在相应段内，也可设置在寄存器中。

5. 编写汇编源程序

在算法和流程图的指导下，逐条编写程序。要求有一定数量的程序说明，以增加程序的可读性。

6. 静态检查

程序编好后，首先要静态检查，查看程序是否具备所要求的功能，是否存在错误。静态检查，认为无错误后，才可上机调试。

1.2.2 简单程序设计实例

例 1.1 求 $y_1=(a+b)-(c+d)$

设数 a,b,c,d 存在内存 NUM 开始的连续四个单元中，运算结果将存在 y_1 单元中，根据问题的要求画出流程图如图 1.4 所示。程序清单如下：

```
;       exam11.asm

dseg    segment
num     db 13h,27h,11h,12h
y1      db ?
dseg    ends
sseg    segment para stack ' stack'
        db 20 dup(?)
sseg    ends
cseg    segment
        assume cs:cseg,ds:dseg,ss:sseg
start   proc far
        push ds
        mov ax,0
        push ax
        mov ax,dseg
        mov ds,ax
        mov bx,offset num          ;设置地址指针
        mov al,[bx]                ;取第一个数
        inc bx
        add al,[bx]                ;加上第二个数
```

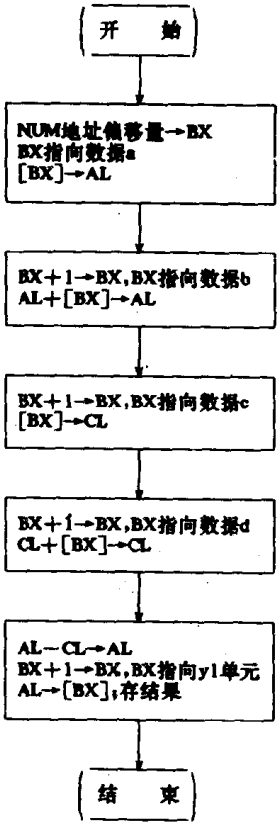


图 1.4 $y_1=(a+b)-(c+d)$ 的程序流程图

```
        inc bx
        mov cl,[bx]                ;取第三个数
        inc bx
        add cl,[bx]                ;加上第四个数
        sub al,cl
        inc bx
        mov [bx],al                ;存结果
        ret
start   endp
cseg    ends
        end     start
```

例 1.2 查表求平方值

内存中自 TABLESQ 开始的 16 个单元连续存放着自然数 0 到 15 的平方值，任给一数 x (0≤x≤15)在 xx 单元中，查表求出 x 的平方值，将结果存入 yy 单元中。

首先在数据段中建立平方表，表的首地址为 TABLESQ，然后用程序找到 x^2 值在平方表中的位置，即计算表地址，表地址＝表起始地址(TABLESQ)＋x，据此可写出如下程序。

```
;       exam12.asm

data    segment
tablesq db 0,1,4,9,16,25,36,49             ;平方值表格
        db 64,81,100,121,144,169,196,225
xx      db 6
yy      db ?
data    ends
stack   segment para stack ' stack'
        db 50 dup(?)
stack   ends
code    segment
        assume cs:code,ds:data,ss:stack
start   proc far
        push ds
        mov ax,0
        push ax
        mov ax,data
        mov ds,ax
        mov bx,offset tablesq              ;建表指针
        mov ah,0
        mov al,[xx]
        add bx,ax                          ;计算表地址
        mov al,[bx]
        mov [yy],al                        ;将答案存入指定单元
        ret
start   endp
```

```
code    ends
        end    start
```

1.2.3 简单程序设计实验

实验 1.1 32 位无符号数乘法

实验要求：

1. 用 16 位乘法指令实现 32 位乘法。

2. 乘数、被乘数和乘积均放在数据段中。

3. 在 PC 机上建立、汇编、连接、调试和运行程序，用 DEBUG 查看运算结果。

提示：

1. 用 16 位指令实现 32 位乘法的算法如图 1.5 所示。

2. 参考程序框图如图 1.6 所示。

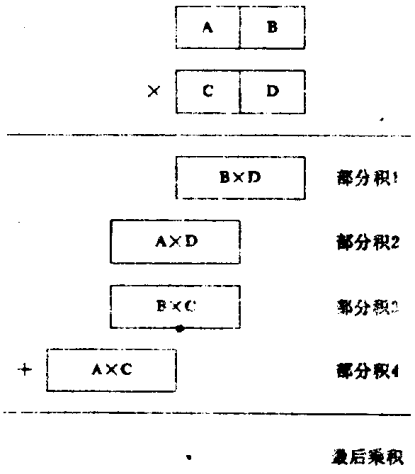


图 1.5 32 位乘法算法

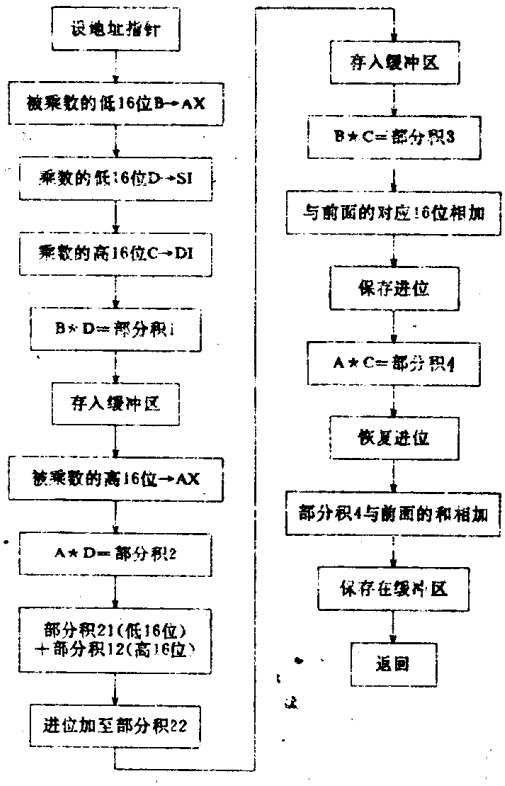


图 1.6 32 位乘法程序流程图

实验 1.2 加减运算

实验要求：

1. 编写程序，计算 3a＋4b－7c→s，已知 a，b，c 分别为内存 DATAA，DATAB，DATAC 单元之内容，s 为内存某单元地址。

2. 在 PC 机上建立、汇编、连接、调试和运行程序，用 DEBUG 查看运算结果。

1.3 循环程序设计

1.3.1 循环程序设计方法

一、循环程序的基本结构

循环程序一般由四大部分组成。

1. 初始化部分

为循环作准备工作，包括建立指针，置计数器，设置其他变量的初值等。

2. 循环体

完成循环的基本操作，核心部分。

3. 修改部分

修改操作数地址等，为下次循环作准备。

4. 控制部分

修改计数器，查看循环控制条件，作循环控制。

二、循环控制的基本方法

1. 计数法

使用条件：循环次数已知

使用形式：正或倒计数法

2. 按条件控制循环

使用条件：循环次数与某些条件有关。条件可以检测。

使用形式：检测，比较，判断。

3. 用开关变量控制分支循环

使用条件：分支规律已知，计数次数或循环条件已知

使用形式：条件判断，转移，正或倒计数法。

4. 用逻辑尺控制循环

第三种循环控制方法适合于编程之前已知分支规律。如果分支规律在编程时无法确定，只有用户运行程序时才能确定，并且分支规律也不固定，这时只能用逻辑尺控制循环。逻辑尺就是判断分支的数据，用户可通过改变逻辑尺来实现不同的分支规律。

1.3.2 循环程序设计实例

例 1.3 计数法控制循环

统计数据段中存放的数据中负元素的个数，并把统计结果存放在数据段结果单元。因为数据的个数已知，故可用计数法控制循环。程序清单如下：

```
;       exam13.asm

data        segment
```

```
d1      db -1,-3,5,6,-9,-8,10          ;定义数组
l       equ $-d1
rs      dw ?                           ;存放负数个数
data    ends
stack   segment para stack ' stack'
        db 100 dup(?)
stack   ends
code    segment
        assume cs:code,ds:data
        assume ss:stack
start   proc far
        push ds
        mov ax,0
        push ax
        mov ax,data
        mov ds,ax
        mov bx,offset d1               ;建立数据指针
        mov cx,l                       ;置计数器初值
        mov dx,0                       ;置结果初值
lop1:   mov al,[bx]
        cmp al,0                       ;用 AND AL,AL 也可以
        jge jus                        ;为正或 0 转移,为负计数器 DX 加 1
        inc dx
jus:    inc bx
        dec cx
        jnz lop1                       ;或 LOOP LOP1
        mov rs,dx
        ret
start   endp
code    ends
        end     start
```

例 1.4 用开关变量控制循环

若在某一数据采集系统中,采集到的前 5 个量,用一种函数进行处理;而采集到的后 7 个量,用另一种函数进行处理。若采集到的数据放在缓冲区 BUFFER 中,处理后的数据放在 BLOCK 缓冲区中。第一种处理函数为子程序 FUN1,第二种处理函数为子程序 FUN2。在程序中设置一个开关(即标志),其初值为 0,控制进行第一种处理;在处理了 5 个数据后,使开关置为 1,控制进行第二种处理,在处理了 7 个数据后恢复初始状态。

```
;       exam14.asm

                name    loop_use_switch
data    segment
buffer  dw 05,05,05,05,05,05,05,05,05,05,05,05
```

```
block    dw 12 dup(?)
count1   equ 5
count2   equ 7
data     ends
stack    segment para stack ' stack'
         db 100 dup(?)
stack    ends
code     segment
         assume cs:code,ds:data,ss:stack
start    proc far
begin:   push ds
         mov ax,0
         push ax
         mov ax,data
         mov ds,ax
         mov es,ax                    ;置段寄存器初值
         mov dx,0                     ;置开关量初值
         mov cx,[count1]              ;置第一种循环次数
         inc cx
         lea bx,buffer
         lea si,block
again:   mov ax,[bx]
         cmp dx,0                     ;开关量是否为 0
         jne anoth                    ;不为 0,转至 ANOTHER
         call fun1                    ;第一种循环
         loop next
         mov dx,1                     ;开关量转为 1
         mov cx,[count2]              ;置第二种循环次数
         inc cx
         jmp again
next:    mov [si],ax
         inc bx
         inc bx
         inc si
         inc si
         jmp again
anoth:   call fun2                    ;第二种循环
         loop next
         ret
start    endp
fun1     proc
         add ax,ax                    ;FUN1 举例,数据乘 2
         ret
```

```
fun1    endp
fun2    proc
        add ax,ax
        add ax,ax                    ;FUN2 举例,数据乘 4
        ret
fun2    endp
code    ends
        end    begin
```

1.3.3 循环程序设计实验

实验 1.3 用逻辑尺控制循环

实验要求：

1. 在例 1.4 中，作如下改进，第 1,2,5,7,10 这几个测量值用第一种处理函数，而第 3,4,6,8,9,11,12 这几个量值用第二种处理函数。编写相应的汇编程序。

2. 在 PC 机上建立、汇编、连接、调试和运行程序，用 DEBUG 查看处理结果。

提示：循环可用一个位串 0011010110110000 来控制，这个位串称为逻辑尺，可放在某一寄存器中，此位串从最高位开始若为 0 则采用第一种函数，若为 1 则采用第二种函数，在程序中每循环一次把此位串左移一位，把控制位移至标志位 C 中，由 C 的值控制转至不同的分支。

实验 1.4 用条件控制循环

实验要求：

1. 编制用牛顿迭代法求一个 16 位无符号数的整数平方根程序。

2. 在 PC 机上建立、汇编、连接、调试和运行程序，用 DEBUG 查看计算结果。

提示：

1. 牛顿迭代法：若 x_1 是数 N 的平方根的近似值，那么 $x_2=(N/x_1\pm x_1)/2$ 是一个更接近的近似值。第一个近似值可用公式(N/200+2)得到。

2. 循环控制条件：要求相邻两次逼近所求得的值之差的绝对值小于或等于 1 就停止循环。

1.4 分支程序设计

1.4.1 分支程序设计方法

分支的实现有多种方法，这里介绍两种基本方法。

一、利用比较转移指令实现分支

利用比较、判断指令如：两数比较指令 CMP，串比较指令 CMPS，串搜索指令 SCAS 等进行判断。根据判断结果，用无条件转移指令 JMP 和各种类型的条件转移指令实现分支。

二、利用跳转表实现分支

跳转表——内存中的一片连续存储单元，连续存放一系列跳转地址、跳转指令或关键字

等。

表首地址——表开始的第一个单元的地址。

表地址——要查找的元素在表中的地址。

偏移量——表地址相对于表首地址的偏差字节数。

1. 根据表内地址分支

若跳转表中存放的是跳转地址，则实现分支的方法如下：

① 根据已知编号计算表地址

表地址＝表基地址＋偏移量(编号 * 每项字节数)

② 分支

JMP [表地址]

2. 根据表内指令分支

若跳转表中存放的是跳转指令，则实现分支的方法如下：

① 根据已知编号计算表地址

表地址＝表基地址＋偏移量

② 分支

JMP 表地址

3. 根据关键字分支

若表中存放的是关键字，则实现分支的方法如下：

① 根据已知编号计算表地址

表地址＝表基地址＋偏移量

② 分支

根据表地址取出相应的关键字的值，再根据关键字的值实现分支。关键字的值即是一个逻辑尺，由它来控制分支。

1.4.2 分支程序设计实例

例 1.5 利用比较转移指令实现分支

在内存区中有一无序列，列的长度存放在第一个字节，要求找出此列中的最小值放在存储单元 minval 中，找出列中的最大值放于存储单元 maxval 中。

```
;       exam15.asm

            name   find_min_max_val
data        segment
buffer      db 10,22,-12,80,-6,-70,-9,127,-10,00,40
minval      db ?
maxval      db ?
data        ends
stack       segment para stack ' stack'
            db 100 dup(?)
stack       ends
code        segment
```

```
        assume cs:code,ds:data,es:data,ss:stack
start   proc far
begin:  push ds
        mov ax,0
        push ax
        mov ax,data
        mov ds,ax
        mov es,ax
        lea bx,buffer
        mov ch,0
        mov cl,[bx]                ;把列的长度送至 CX
        inc bx                     ;BX 指向列中的第一个元素
        dec cx
        mov al,[bx]                ;把列中的第一个元素取至 AL 中
        mov [minval],al
        mov [maxval],al
        inc bx                     ;BX 指向列中的第二个元素
        dec cx                     ;调节循环次数
again:  mov al,[bx]                ;取出列中的各个元素
        cmp al,[minval]            ;与最小值单元中的值相比较
        je next                    ;相等转至 NEXT
        jg aler                    ;列中元素大,转至 ALER
        mov [minval],al            ;把小的值送至 MINVAL
        jmp next
aler:   cmp al,[maxval]            ;与最大值单元中的值相比较
        jle next                   ;列中元素的值小,转至 NEXT
        mov [maxval],al            ;把大的值送至 MAXVAL
next:   inc bx
        dec cx
        jne again
        ret
start   endp
code    ends
        end     begin
```

例 1.6 根据表内地址分支

某工厂有几种产品的加工程序 R0,R1,…,Rn,分别存放在 PR0,PR1,…,PRn(n<256)为首地址的内存区域中,而这几个首地址的偏移量连续存放在以 BASE 为地址的转移表内。如表 1-2 所示。

已知产品编号存放在 num 单元,要求编写根据产品编号转至相应处理程序的程序。

表 1-2 转移表

BASE	PR0
	PR1
	PR2
	PR3
	⋮
	PRn
PR0	
	…
PRi	
	…
PRn	

```
;       exam16.asm
```

```
dseg    segment
base    dw pr0,pr1,...,prn
num     db 5
dseg    ends
sseg    segment para stack ' stack'
        db 20 dup(?)
sseg    ends
cseg    segment
        assume cs:cseg,ds:dseg,ss:sseg
begin   proc far
        push ds
        xor ax,ax
        push ax
        mov ax,dseg
        mov ds,ax
        mov al,num              ;i->AL
        mov ah,0
        add ax,ax               ;2*i->AX
        mov bx,offset base      ;建表指针
        add bx,ax               ;处理程序入口地址的地址->BX
        mov ax,[bx]             ;处理程序入口地址->AX
        jmp ax
        ret
begin   endp
cseg    ends
        end     begin
```

1.4.3 分支程序设计实验

实验 1.5 利用比较转移指令实现分支

实验要求：

1. 编写程序，计算 $Fx=\begin{cases}0 & (x>5)\\ 1-x & (x\leqslant 5)\end{cases}$

2. 在 PC 机上建立、汇编、连接、调试和运行程序，用 DEBUG 查看运算结果。

实验 1.6 利用表内指令分支

某一监控程序中有 12 个命令，分别以字母 A,B,C,D,E,F,G,H,I,J,K,L 表示，这 12 个命令对应于 12 个处理程序，分别由 12 个跳转指令来实现转子功能。这 12 个跳转指令组成了一个跳转表，如图 1.7 所示。已知跳转表中每三个单元存放一条转移指令，键入的命令字是以 ASCII 码的形式存在 NUM 单元中。

实验要求：编程实现根据命令字转至相应的处理程序。

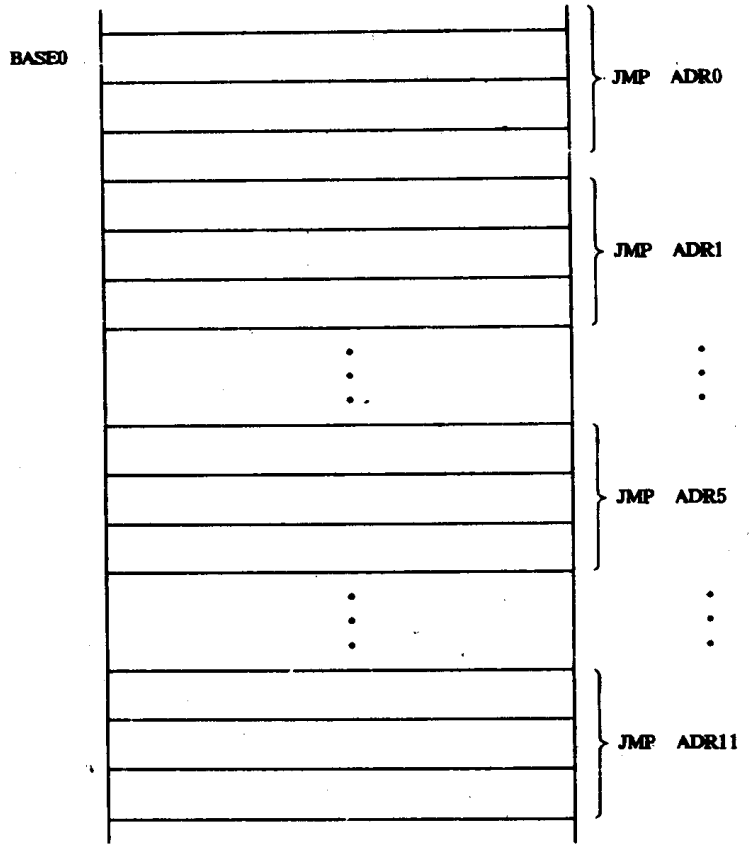


图 1.7 命令键跳转表

1.5 子程序设计

1.5.1 子程序设计方法

一、参数的传递方法

子程序和调用程序之间的参数传递，一般有三种方法：

(1) 用寄存器传递参数：当参数较少时，可把参数放在寄存器中，这样主程序和子程序均可使用。

(2) 用参数表传递参数：当参数较多时，可把要传递的参数均放在参数表中，然后把表的首地址传到子程序中，子程序通过地址表取得参数。

(3) 用堆栈传递参数：当参数较多并且子程序有嵌套，递归调用的情况时，主程序将参数推入堆栈，子程序将参数从栈中弹出。

二、工作单元的保护

如果在子程序中要用到某些寄存器或存储单元，为了不破坏原有信息，要将它们的内容

推入堆栈，存入另外一些空闲的存储单元或存入某些目前不用的寄存器中加以保护。保护可以在子程序中实现，也可以在主程序中实现，最好是在子程序的开始安排一段保护程序，用以对它所用到的寄存器或存储单元的内容加以保护，在子程序结束前，再将有关内容恢复，这样可防止粗心的用户忘记在调用前安排保护引起的错误。

1.5.2 子程序设计实例

例 1.7 十六进制数转换为十进制数

该程序把从键盘输入的 0～FFFFH 的十六进制数转换为十进制数并从屏幕上显示出来。转换过程由两个子程序完成，子程序 hexibin 把键盘输入的十六进制数转换成二进制数。子程序 binidec 把二进制数转换成十进制数并在屏幕上显示。参数采用寄存器传递方式。

```
;     exam17.asm

              ;把由键盘输入的十六进制数转换成十进制数并在屏幕上显示
data          segment
display       equ 2h                          ;显示器输出
key_in        equ 1h                          ;键盘输入
doscall       equ 21h                         ;DOS 中断类型号
data          ends
stack         segment para stack ' stack'
              db 100 dup(?)
stack         ends
hexidec       segment
main          proc far
              assume cs:hexidec,ds:data,ss:stack
start:        push ds                         ;开始执行地址
              sub ax,ax
              push ax
              mov ax,data
              mov ds,ax
              mov ax,stack
              mov ss,ax
              call hexibin                    ;键盘输入十六进制数转换为二进制数
              call crlf
              call binidec
              call crlf
              ret
main          endp                            ;主程序结束
hexibin       proc near                       ;子程序 HEXIBIN,把键盘输入的十六进制
                                              ;数转换为二进制数
              ;mov bx,0
              ;从键盘输入数字,并转换成二进制数
newchar:      mov ah,key_in                   ;键盘输入 DOS 调用
```

```
        int doscall
        ;判断是否为数字(0-9)
        sub al,30h                  ;ASCII 码变为二进制
        jl exit                     ;判断是否为数字(0--9)
        cmp al,0ah
        jl add_to                   ;是数字转移
        ;判断是否为字母(a-f)
        sub al,27h                  ;ASCII 码变为二进制
        cmp al,0ah
        jl exit                     ;不是字母 A-F 转移
        cmp al,10h
        jge exit                    ;不是字母 A--F 转移
        ;是十六进制数字,则加到 BX 中
add_to: mov cl,4                    ;左移位数
        shl bx,cl                   ;BX 左移 4 位
        mov ah,0
        add bx,ax                   ;输入数字存入低四位
        jmp newchar
exit:   ret
hexibin endp
binidec proc near                   ;该子程序把 BX 中的二进制数转换成
                                    ;十进制数,并在屏幕上显示
        mov cx,10000d               ;求万位数
        call dec_div
        mov cx,1000d                ;求千位数
        call dec_div
        mov cx,100d                 ;求百位数
        call dec_div
        mov cx,10d                  ;求十位数
        call dec_div
        mov cx,1d                   ;求个位数
        call dec_div
        ret
dec_div proc near                   ;实现 BX/CX
        mov ax,bx
        mov dx,0
        div cx
        mov bx,dx                   ;余数放入 BX 中
        mov dl,al                   ;商放入 DL 中
        add dl,30h                  ;商转换成 ASCII 码
        mov ah,display              ;显示商
        int doscall
        ret
```

```
dec_div     endp
binidec     endp
crlf        proc near
            ;输出换行和回车
            mov dl,0ah                    ;换行
            mov ah,display
            int doscall
            mov dl,0dh                    ;回车
            mov ah,display
            int doscall
            ret
crlf        endp
hexidec     ends
            end     start
```

例 1.8 十六进制数转换为 ASCII 码

把内存中的字变量 NUMBER 的值，转换为 4 个用 ASCII 码表示的十六进制数码串，串的起始地址为 STRING。子程序 binhex 采用堆栈传送参数，子程序 hexd 采用寄存器传送参数。

```
;     exam18.asm

            name    use_subr_exam1
data        segment
num         dw 25afh
string      db 4 dup(?),' $' ,0dh,0ah
data        ends
stack       segment para stack ' stack'
            db 100 dup(?)
stack       ends
code        segment
            assume cs:code,ds:data,es:data,ss:stack
begin:      mov ax,data
            mov ds,ax
            mov es,ax
            lea bx,string                 ;建立地址指针
            push bx                       ;BINHEX 的入口参数进栈
            push num
            call binhex
            lea dx,string
            mov ah,9
            int 21h
            mov ah,4ch
            int 21h
binhex      proc
```

```
        push bp                     ;入栈保护 BP
        mov bp,sp
        push ax                     ;保护现场
        push di
        push cx
        push dx
        pushf
        mov ax,[bp+4]               ;入口参数 NUM 弹出到 AX 中
        mov di,[bp+6]               ;入口参数 BX(地址指针)弹出到 DI 中
        add di,length string-1      ;DI 指向十六进制数码串中的个位
        mov dx,ax                   ;NUM 值送 DX 保存
        mov cx,4                    ;转换数据的位数送 CX
again:  and ax,0fh                  ;屏蔽高位
        call hexd                   ;把十六进制数字转换成 ASCII 码
        std                         ;地址减量
        stosb                       ;存转换结果
        push cx
        mov cl,4
        shr dx,cl                   ;把 NUM 的高位数移到 AX 的最低四位以备转换
        mov ax,dx
        pop cx
        loop again                  ;循环转换 NUM 的各位数字,直到转换完
        popf                        ;恢复现场
        pop dx
        pop cx
        pop di
        pop ax
        pop bp
        ret 4                       ;返回并废除入口参数
binhex  endp
hexd    proc
        cmp al,0ah
        jl addz                     ;是数字 0-9 转移
        add al,' a' -' 0' -0ah      ;把 0AH-0FH 转换成 A-F 的 ASCII 码
addz:   add al,' 0'                 ;转换成 ASCII 码
        ret
hexd    endp
code    ends
        end     begin
```

1.5.3 子程序设计实验

实验 1.7 十进制数转换为十六进制数

实验要求：

1. 从键盘中取得一个十进制数，然后把该数以十六进制形式显示出来。

2. 采用子程序结构。

3. 在 PC 机上建立、汇编、连接、调试和运行程序。

提示：参考例 1.7 的程序设计方法，用一个子程序 decibin 实现从键盘取得十进制数，并把它转换为二进制数，用子程序 binihex 把此二进制数以十六进制数的形式在屏幕上显示出来。参数可采用寄存器传送方式。

实验 1.8 数的阶乘

实验要求：

1. 编程序计算变量 NUMB 的阶乘，把结果存入变量 FNUMB。变量 NUMB 的值大于 0 小于 9。

2. 在 PC 机上建立、汇编、连接、调试和运行程序，用 DEBUG 查看运算结果。

提示：

1. 阶乘的计算公式

$$n! = \begin{cases} n*(n-1)! & \text{当 } n>0 \text{ 时} \\ 1 & \text{当 } n=0 \text{ 时} \end{cases}$$

2. 用子程序的递归调用实现，参数采用堆栈传递方法。

1.6 80386、80486 汇编程序设计

1.6.1 80386、80486 汇编程序设计方法

80386 和 80486 增加的许多功能，目前使用的 DOS 并不支持，因此在使用 DOS 的 386 和 486 机器上只体现出其运行速度比 8088 快，而保护方式和虚拟方式，DOS 均不支持。

但对于编程来说，在目前 DOS 下，可以在 80386 和 80486 机器上使用一些新增加的指令（除保护态下的那些指令外），如位扫描（BSF 和 BFR）、位设置（BT、BTC、BTR 和 RTS）、有符号和零扩展的传递指令（MOVSX 和 MOVZX）、设置条件字节指令（SET 条件）、双精度移位指令（SHLD 和 SHRD）。

也可使用一些加强了的功能指令，如可使用 32 位寄存器的指令，如整数乘（IMUL）、转换指令（CWDE 和 CDQ）、串指令（CMPSD、LODSD、MOVSD、SCASD、STOSD、INSD、OUTSD）和 32 位堆栈操作指令（PUSHAD、POPAD、PUSHFD、POPFD 和 IRETD）。

可以使用 32 位寄存器来进行计算，如可以不用多个寄存器而进行双字长的加减运算，也可以进行 64 位整数乘除运算，也可以用 32 位寄存器来指向 16 位的段，对 8088 和 80286 只能用 BX、BP、DI 和 SI 作为指针来间接地指向存储器操作数。80386 若使用 16 位寄存器时，也有此限制，但是当使用 32 位寄存器时，任何一个 32 位通用寄存器均可用作间接访问存储器操作数的指针，但要确保这些 32 位通用寄存器中存放的是 16 位的地址指针。

1.6.2 80386、80486 汇编程序设计实例

例 1.9 32 位数加法

用 32 位寄存器指令实现 32 位数据加法。

```
;     exam19.asm

          .386          ;使用 80386 指令系统
sseg      segment use16 stack          ;堆栈段以 16 位寻址
          db 100 dup(?)                ;设有 100 个字节的堆栈空间
sseg      ends
dseg      segment use16                ;数据段以 16 位寻址
buf       dd 12345678h,35af8657h,?
dseg      ends
cseg      segment use16                ;代码段以十六位寻址
          assume cs:cseg,ds:dseg,ss:sseg
start     proc far
begin:    push ds                      ;为返回 DOS 作准备
          mov ax,0
          push ax
          mov ax,dseg                  ;置 DS 初值为 DSEG 段的起始地址
          mov ds,ax
          mov ax,sseg                  ;置 SS 初值为 SSEG 段的起始地址
          mov ss,ax
          mov sp,100                   ;使堆栈指针 SP 指向堆栈底
          mov eax,buf                  ;取第一个数->EAX
          add eax,buf+4                ;与第二个数相加,结果->EAX
          mov buf+8,eax                ;结果送 BUF 的第三个双字单元中
          ret                          ;返回操作系统
start     endp
cseg      ends
          end     begin
```

例 1.10　32 位数的乘法

用 32 位寄存器指令进行 32 位乘法运算,并将结果显示在屏幕上。

```
;     exam110.asm

          dosseg
          .model small                 ;小模式
          .386                         ;使用 80386 指令系统
          .stack                       ;堆栈段
          db 256 dup(?)
          .data                        ;数据段
num1      dd 12345678h                 ;32 位被乘数
num2      dd 9abcdef0h                 ;32 位乘数
answer    dq ?                         ;存放 64 位乘积的四字
disnum    db 16 dup(0),'  $'           ;存放乘积的 ASCII 码
          .code
myproc    proc far
```

```
begin:  push ds
        sub ax,ax
        push ax
        mov ax,@data
        mov ds,ax
        mov eax,num1
        mul num2                        ;作两个 32 位数的乘法
        mov dword ptr answer[0],eax     ;乘积低 32 位保存在前四个单元
        mov dword ptr answer[4],edx     ;乘积的高 32 位保存在后四个单元
        ;在屏幕上显示 64 位乘积的程序段
        mov cx,16                       ;乘积要转换成 16 个 ASCII 码
        lea bx,answer[0]                ;得到存放乘积的首地址单元
again:  mov ax,[bx]                     ;AX 中为 16 位数据
        and ax,0fh                      ;只保留低四位
        add al,30h                      ;变成 ASCII 码
        cmp al,39h                      ;是字母吗?
        jl ascii                        ;若是数字,则转
        add al,07h                      ;将字母调整成 ASCII 字符
ascii:  mov si,cx
        mov disnum[si],al               ;保存 ASCII 字符,数字
        ror dword ptr answer,4          ;把待转换的下一个数移到最低位
        cmp cx,8
        jnp lp                          ;低 32 位乘积转换完否?
        mov eax,dword ptr answer[4]     ;若转换完,则将高 32 位乘积送到原
                                        ;低 32 位乘积地址
        mov dword ptr answer[0],eax
lp:     loop again
        lea dx,disnum
        mov ah,9                        ;DOS 显示串功能调用,显示串地址在 DX 中
        int 21h
        ret
myproc  endp
        end     begin
```

1.6.3 80386、80486 汇编程序设计实验

实验 1.9 统计 AX 中位值为 1 的个数

实验要求:

1. 编程统计存入 AX 寄存器中的二进制数位值为 1 的个数,统计结果存放在变量 one 中。

2. 用 80386 位操作指令 BT 来测试 AX 中的每一位是 0 还是 1。

3. 在 386 机器上建立、汇编、连接、调试和运行程序。

1.7 汇编语言与高级语言的接口技术

在微机应用系统设计中，既要考虑硬件的实现过程，又要结合硬件进行相应的软件设计。一些功能用硬件实现起来很麻烦，可以用软件实现。应用系统软件主要由高级语言和汇编语言来编写，对于繁杂的计算、数据处理和用户界面的设计，若采用汇编语言来实现，就会感到十分困难，一般用 C 语言、BASIC 等高级语言来实现，对于硬件驱动程序，若采用高级语言就很难实现，一般采用汇编语言来编写。因此，对于一个程序设计人员来说，就要面对混合语言程序设计这样一个实际问题需要去解决。

1.7.1 混合语言程序设计环境要求

混合语言程序设计的环境是什么，各语言之间有什么不同，如何在这些不同点上建立语言程序间的接口，这是混合语言程序设计首先要了解的问题，因为以各种源语言编制的程序最终都要在同一个运行环境下执行，那么，它们之间的关联是什么？在什么时候和如何来进行语言程序间的调用？这是程序设计人员首先应解决的问题。

软件接口的一个重要类型是程序间的接口，统称程序接口。这种程序接口方式一般以调用型方式为主，由于各种语言在结构等细节上规则不同，必须考虑程序模块之间的接口，了解造成不同的原因并找出对应解决办法。这些不同点除了语言自身的结构与规则不同外，在程序调用时一般表现在如下一些方面。

1. 命名约定和使用的不同

各个语言在编译时对变名的处理以及对目标文件中标识符名的生成方式各有不同。以 MS 系列语言为例，FORTRAN 语言将截取程序中所有变名（如变量名、常数名、过程名、函数名等标识符）的前 6 个字符作为目标名。C 语言和 Pascal 语言一般截取名字前的 8 个字符作为目标名，此外，C 语言还自动在名称前方加上一个下划线“_”，如使 fact 变成_fact。汇编语言可以接收承认 31 个字符为目标名，而 BASIC 语言的目标名则最多可以容纳 40 个字符，并在编译时自动消除名字中的类型描述字符（如%，&，# 等）。所以在混合语言编程中必须考虑命名的约定问题。

2. 参数传递的方式不同

两个程序之间的参数传递必须配合默契，方能得到正确的结果，才能真正达到数据共享。所以应当了解每一种语言在其缺省情况下的参数传递方式是什么，它可以允许几种传递方法，需要采用什么样的技术才可以改变参数的传递方式等等。

另外一个与参数传递有关的问题是各个语言对堆栈的操作。由于今后大部分数据和参数是通过堆栈传送的，所以必须了解各语言的堆栈结构，生成方式，参数入栈方法等等。我们知道，对于 BASIC（编译型），FORTRAN 和 Pascal 语言，其参数压栈的顺序是与参数在参数表中出现的顺序相同的，即顺序从左到右，而 C 语言却刚刚相反，顺序从右到左，例如，过程 fact(a,b,c,d)的参数压栈结果就如图 1.8 所示。图中左边对应 BASIC（编译型）等语言的压栈情况，右边对应 C 语言参数的压栈情况。而且，参数在堆栈中所占的字节与调用类型，参数类型等都有关系，也必须弄清楚，因为它涉及到堆栈的恢复和正常返回。此外，还有字符串，数组等特殊数据的定义，说明及传递，都必须按某种约定进行，使两种语言都按彼此兼容的

方式对同一数据类型或者结构进行操作和处理。

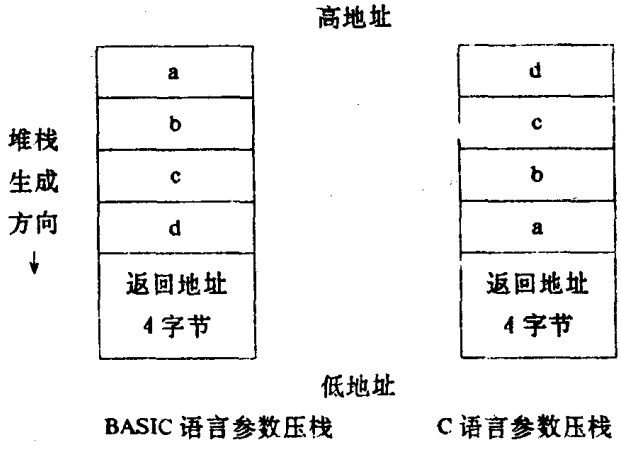


图 1.8 不同的参数压栈结构

3. 调用方式与实现

混合语言编程的程序接口通常都包含了一个调用的过程，即对其他的外部程序过程进行调用。这些外部过程既可以是用本语言编写的，也可以是以任何一种语言编写的。它可以是一个函数调用，一个过程调用或是一个子程序调用。

混合语言涉及用多种语言书写的多个程序模块，这些程序模块不是采用同一个编译程序，而是分别采用不同的编译程序或者汇编程序来编译或汇编的。

1.7.2 怎样编写适合于混合语言接口的汇编程序

众所周知，汇编语言具有存储段结构，在高级语言编译所产生的目标文件中，也包含有存储段信息，以便连接程序连接。本节主要以宏汇编 MASM 为例，讨论混合语言接口中汇编程序的编写。

要想用连接程序 LINK 把一个汇编程序过程连接到由另一种语言所写的主调程序中，两种模块必须使用相同的(或相兼容的)存储段，而且双方都按照一定的约定进行了适当的说明。汇编语言程序所要遵守的约定为：

(1)在汇编语言源程序开头使用指示字. MODEL，这个指示字将自动产生适当类型的调用和返回情形。比如，对小型(small)和紧凑型(compact)存储模式将自动产生近程(段内 NEAR)调用，而对其他存储模式，如巨型(huge)，大型(Large)，中型(medium)将自动产生远程(段间 FAR)调用。如果因汇编系统的版本较早而不支持汇编指示字. MODEL，可把汇编过程都说明成 FAR，对小型或紧凑型的 C 语言调用方式，也可以说明成 NEAR。

(2)分别用汇编指示字. CODE 和. DATA 对汇编过程中的代码段和数据段给予说明。其中，数据段说明可以没有，即不定义数据，但重要的是要有代码段。如果版本较低，不支持此指示字，可以按照传统的宏汇编格式分别用 segment，group 和 assume 指示字写出，这样要复杂一点，但结果是一样的。建议不要使用传统的宏汇编格式，否则你将会遇到不必要的烦恼。

(3)一个被调用的汇编过程名必须用指示字 PUBLIC 说明成一个公共块，命名这个模块可以被其他的模块使用和访问。同理，如果想让数据也成为可被其他程序模块使用的公共数据，也必须把它说明成 PUBLIC。

(4)由程序访问全局变量,数据或者过程都必须说明成 EXTRN,一般是把 EXTRN 指示字引出的说明放到所有的段定义之外,不过,对近程(段内)数据也可以放在数据段内。这样,汇编程序模块的结构就形如图 1.9 所示。

MASM V5.0 以后版本	MASM V5.0 以前版本
.MODEL	Data segment public 'DATA'
.DATA	msg dw …
msg dw …	……
……	Data ends
.CODE	Code segment
.	assume cs:code …
xx proc far	xx proc far
…	…
xx endp	xx endp
…	…
end	Code ends
	end

图 1.9 汇编程序一般结构

在构造好了汇编程序结构之后,在汇编程序中还应为其他高级语言调用做如下的准备工作:

(1)汇编过程的进入

汇编过程的进入一般以二条指令开始,即

```
push bp
mov bp,sp
```

这两条指令使寄存器 BP 成为一个区域指针,用来访问参数和局部数据。这些数据都存放在堆栈中,所以,BP 也成为一个指向堆栈中各元素的指针,正是这个 BP 成为了参数传递的关键。

为什么要用 BP 代替 SP 而不直接使用堆栈指针 SP 呢?第一,SP 不能用于对参数和数据进行访问,因为它不是一个索引或者基址寄存器。第二,如果利用 SP,而 SP 的内容每次都将随着程序中的 push 和 pop 指令的执行而改变,以 SP 作为参考访问数据容易发生错误。所以,必须用另一个寄存器作为基址来对堆栈段寻址,这个寄存器就是 BP。基址寄存器 BP 不受 push 和 pop 等堆栈操作的影响,在整个过程间保持一个常数值。从而,所传递的参数都以 BP 作为一个参考点,参数寻址都与 BP 保持一个固定的位移量。

上述的第一条指令是把 BP 的原值保护起来,这是因为在当前过程终止后调用过程需要恢复其原值。第二条是把堆栈指针的值赋给 BP 以便一进入被调用过程就获得堆栈指针,

作好参数访问准备。这样，我们就可以间接地用 BP 来寻址堆栈段了。

(2)局部数据分配(可选择)

在程序中，局部变量(数据)也被称为动态变量，堆栈变量或者自动变量。当编程者要进行局部数据分配时，汇编程序过程采用与高级语言相同的技术来实现局部数据分配和存取。为了建立这样一个局部数据空间，可以通过 SP 的值的减少，就可以在堆栈中的局部数据保留一个空间，而这个空间必须在汇编过程结束时予以恢复。即进入汇编后用：

```
push bp
mov bp,sp
sub sp,space
```

这里，space 是以字节计算的局部数据区的总大小，对局部变量和数据的访问则可以用一个以 BP 为基准的固定的负位移量值来进行。例如：

```
push bp
mov bp,sp
sub sp,4
  ⋮
mov WORD PTR[bp-2],0
mov WORD PTR[bp-4],0
```

上述例子中用了两个局部变量，每一个都是两个字节大小，所以，保留了 4 个字节的局部数据空间，被保留的空间是 SP－4。然后，这两个变量被初始化为 0，它们不需要用汇编系统的任何指示字进行形式上的说明，所以，编程者必须人为地对这些变量进行跟踪。当然，如果无需分配局部数据区域，这一步是可以不进行的。

(3)现场及相关寄存器

由高级语言调用的汇编过程需要进行现场保护，一般来说是保护有关的重要寄存器的前值，如寄存器 SI，DI，SS 和 DS 的值(BP 的值在进入汇编过程时已经保护了)。因此，应该把汇编过程要改变的寄存器的值压入堆栈，当然，如果编程者明确知道某些寄存器的值不会因汇编过程的执行而改变，也可以不压入堆栈，不进行保护。

在高级语言调用汇编过程时，最好的办法是在被调用的汇编程序中首先设置基址指针 BP，其次设置局部数据区(如果需要的话)，然后再保护有关的寄存器的值，如下：

```
push bp
mov bp,sp
sub sp,4
push si
push di
...
```

被保护的寄存器在汇编过程结束前应予恢复。

(4)对参数的访问

这是一个极其重要的问题。一旦我们建立了汇编过程的区域指针，也对所需要的局部数据开辟和保留了空间，要保存的寄存器值也入了堆栈，那么，就可以编写汇编过程的真正的主体了。为了利用指令来访问参数，我们先来看看在过程调用时堆栈的结构情况。经过上述

几步产生的堆栈结构如图 1.10 所示。

其中参数和返回地址是由调用程序执行调用指令压入堆栈的。这个返回地址可以是二个字节(对段内调用),也可以是四个字节(对段间调用),由 SP 指向这个地址。

从堆栈结构图上可以看出,要在堆栈中找到参数,可通过查表确定,表基地址为 BP,位移量的计算如下:

BP 的位移量=2+返回地址大小+参数 x 与 BP 间的参数所占的总字节数

例如:一个以段间调用的过程要接收一个参数,该参数为二字节,且该参数与 BP 间无其他参数,那么,这个参数的位移量就是:

参数位移量=2+返回地址字节数=2+4=6

这样,变量就可以通过对 BP 加上位移量并用下述指令进行访问:

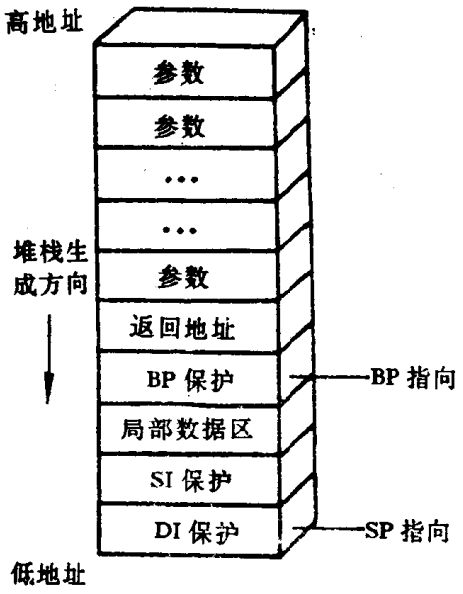


图 1.10 堆栈生成结构

```
mov bx,[bp+6]
```

如果编者对每个参数的位移量都已确知,就可以在其汇编语言源码中用字符串等价或者结构类型进行说明,使参数可以利用一个单个的标识符名称来进行访问。比如,上例中位于 bp+6 位置的参数就可以很方便地访问,也可以用等价说明写成:

```
arg1 EQU [bp+6]
```

来访问该参数,以后在程序指令中凡是有这个参数出现的地方都用 arg_1 符号代替,这既有助记的好处又便于今后对参数的增删和修改。

一般来说,高级语言总是先将段地址压入堆栈,再将位移量压入堆栈。类似地,当参数所占字节数大于二字节时,也总是先将高地址部分压入堆栈,再将低地址部分压入堆栈。

如果是汇编语言调用高级语言程序时,也需把每一个参数都压入堆栈,并按照高级语言的调用约定对堆栈进行访问。对于长型参数,也总是把段地址或高位字段首先压入堆栈,并注意地址所占的字节和不同类型参数所占用的字节数。此外,执行调用指令时,除非高级语言程序采用 small 模式编译,一般都采用段间 FAR 调用。

(5)结果和值的返回(可选择)

在 PC 系列机上,当返回值的数据类型是简单类型时(即不含数组或者结构等类型),并且返回值长度不超过四个字节时,接收约定是相同的。一般具有如下约定:

数据大小	返回值所放寄存器
1 字节	AL
2 字节	AX
3 字节	高端地址(或段地址)在 DX
	低端地址(或位移量)在 AX

不过,当返回值超过四个字节时,由 BASIC 或者 C 语言调用的过程必须为这些返回值分配一个存储空间,然后把它的地址放在寄存器 DX:AX 中。为返回值产生空间的一种方便的方

法是在数据段中进行简单的说明。

(6)退出汇编程序过程

在结束汇编过程而退出时,需要执行下列几步:

① 如果在汇编过程开始时保存了对应的寄存器值,那么,在汇编过程结束前,必须以它们入栈时的相反顺序将它们一一弹出堆栈。

② 如果在汇编过程开始时分配了局部数据空间,那么,在汇编过程结束时,必须用指令 mov sp,bp 恢复 sp 的原来内容。

③ 用指令 pop bp 恢复 bp 的原值,这一步至关重要,不可缺省。

④ 最后用指令 RET 返回调用程序。但要注意,如果调用程序是 BASIC,FORTRAN 或者是 Pascal 语言程序,那么,应该使用 RETn 的指令形式去恢复堆栈。如果调用程序是 C 语言程序,则只用 RET 指令,调用模块会自动完成堆栈恢复。

最简单的退出恢复堆栈是:

```
pop bp
ret
```

这里,开始没有保护任何寄存器,也没有分配局部数据空间,且主调程序是 C 语言。

稍复杂的退出过程:

```
pop di          ;恢复被保护的寄存器
pop si
mov sp,bp       ;去掉局部数据空间
pop bp          ;恢复原 bp 值
ret 6           ;恢复(跳过)6 字节参数后退出
```

这个退出序列既要对原被保护的寄存器 SI 和 DI 予以恢复,又要消去已分配的局部数据区。如果过程采用的是非 C 语言的调用约定,汇编过程必须用 ret n 指令来退回堆栈中参数所占的 6 个字节。

1.7.3 汇编语言与 C 语言之间的接口

要实现汇编语言与 C 语言之间的接口,必须遵循三个约定,即命名约定、调用约定和参数传递约定,并在彼此程序中作对应的说明。本节主要以 Turbo C、C++和 Borlandc、C++为例,讨论宏汇编 MASM 与 C 语言的接口方法。

一、C 语言对汇编程序的调用

1. C 语言对汇编程序调用方式

C 语言程序中的接口处理方法如下:

(1)命名约定

在 C 语言源程序中调用一个外部汇编过程时,直接使用该过程名字而不加下划线。比如,在汇编语言程序中有一个被调用的过程名为_demo,在 C 程序中可直接用 demo( )对其产生调用。因为 C 程序编译后,此调用名自动转换成_demo,自然与汇编过程名相符。

此外，C 语言规定名称的有效长度为 8 个字符，因此，当外部汇编过程名多于 8 个字符时，C 语言程序对它的调用只取前 8 个字符（不包括下划线）作为调用过程名，例如：变名_abcdefghi 在调用时只取 abcdefgh 8 个字符。

(2)调用约定

在 C 语言程序中对所要调用的外部过程（含汇编过程及变量等）均采用标准的 extern 关键字予以说明。这个 extern 说明必须放在主调程序调用这个外部过程之前，一般放在各函数体外部。用这个 extern 可以对任何外部过程、函数、变量，以及其他数据类型进行说明。在主调程序中，一旦遇到已被说明过的变量名或者过程名，系统就到其他程序模块中去找寻。extern 语句的说明形式如下：

extern 返回值类型　名称(参数类型表)

这里，返回值类型是对函数过程而言，对变量则是指变量类型。这些类型是 C 语言所允许的任何数据类型，如 int，char，float，void 或者指针 int * 类型等。如果省略此项，缺省默认为 int 型。名称是对应的函数过程名或者变量名等，它应当符合命名约定。参数类型表包含了 C 语言要向另一个程序模块传送参数的类型，如果无参数传送，可以空缺。

例如：extern short thing(int,short)；在用 extern 语句进行说明后，就可以在 C 语言程序中直接使用了。在进行函数过程调用时，如果要进行参数传递，所传递的实际参数不仅在数量上要与 extern 语句中参数类型表中的参数个数一致。而且，参数的类型也要一一对应。

(3)参数传递约定

如果不进行参数传递，则外部说明和调用时都予以空缺，例如：

```
extern thing( );
   ⋮
thing( );
```

如果参数是传值传送，可以直接写出实际参数。如果是传址传送，则在 extern 说明中，将参数类型说明成指针型，并在放实参时给出参数的地址。注意，在 C 语言中数组总是以传址方式进行的，所以，数组传送时可以不用一元算符 &。例如：

```
extern int.thing(int *);
…
int n=10;
…
thing(&n);
```

被 C 语言调用的汇编语言程序应完全按照上一节提出的要求予以编写，根据 C 语言的特点还应遵循如下约定：

(1)段的组合约定

C 编译有其自己产生模块名，段名，组名，类别名和组合类型的办法，汇编语言也有其自己产生模块名，组名，类别名和组合类型的办法。为了使两种语言产生出来的模块能正确地互相组合起来，在写源程序时，就应该加以认真的考虑。比较起来，能否正确地组合起来更多的是取决于汇编语言程序，因为汇编语言提供了更多更灵活的控制这些参数的办法。

C 语言要求的汇编语言的一般格式如表 1-3 所示。

表 1-3　汇编语言文件格式

```
code        SEGMENT      BYTE  PUBLIC ' CODE'
            ASSUME       CS:code,DS:dseg
            ……代码段……
code        ENDS

dseg        GROUP        _DATA,_BSS
data        SEGMENT      WORD  PUBLIC ' DATA'
            ……初始化数据段…
data        ENDS

_BSS        SEGMENT      WORD  PUBLIC ' BSS'
            ……非初始化数据段…
_BSS        ENDS

            END
```

为了使这个汇编语言程序模块能与C语言程序模块组合起来，这个格式中的“代码段名”code、“组名”dseg、“数据段名”data 应该与 C 编译产生的“代码段名”code、“组名”dseg、“数据段名”data 相一致。而 C 编译产生的这些名字又与编译时的内存模式紧密相关，如表 1-4 所示。

表 1-4　标识符替换和内存模式

模式	标识符	可替换代码和指针
微，小	code=_TEXT	Code:　DW　_TEXT:xxx
	data=_DATA	Data:DW DGROUP:xxx
	dseg=_DGROUP	
紧凑	code=_TEXT	Code:DW　_TEXT:xxx
	data=_DATA	DATA:DD DGROUP:xxx
	dseg=_DGROUP	
中	code=filename_TEXT	Code:DDxxx
	data=_DATA	Data:DD DGROUP:xxx
	dseg=_DGROUP	
大	code=filename_TEXT	Code:DDxxx
	data=_DATA	Data:DD DGROUP:xxx
	dseg=_DGROUP	
巨	code=filename_TEXT	Code:DDxxx
	data=filename_DATA	Data:DDxxx

在这个表中的标识符 code，data 和 dseg 有特定的替换方法，这取决于所使用的内存模式。表中的文件名是编译时指定的模块名，汇编语言中，是通过伪指令 NAME 来指定的。

注意对 huge 模式，没有_BSS 段，GROUP 定义也被完全删除。一般来讲，_BSS 是可选择

的，如果需要使用它，只能自己定义。

得到一个汇编语言样例的最好方式，是将一个空的 C 语言程序编译到 ASM(使用 TCC 选项－S)。关于这个问题将在后面讨论。

除使用上述标准段指令外，还可以用简化的段指令. Code、. DATA 和. DATA? 来定义代码段、初始化数据段和非初始化数据段，这时汇编语言文件格式为：

```
.MODEL  内存模式
.CODE
…代码段…
.DATA
…初始化数据段…
.DATA?
…未初始化的数据段…
```

由此可见，用简化段指令会给你带来极大的方便。

如果 C 语言程序以巨型(huge)、大型(large)或者中型(medium)存储模式编译，被 C 调用的汇编过程应说明成 NEAR。当然，如果采用较新的汇编语言版本，如 MASM V5.0 或 V5.1，则可以不考虑对每一过程进行说明，只需直接采用汇编指示字. MODEL 说明，它隐含了上述操作。

(2)变量和常数定义约定

在准备与 C 语言程序连接的汇编语言程序里，定义变量和常数是件很容易的事，与普通汇编程序差不多，但也有两点值得考虑。第一，如果不给变量赋初值，则可以把这些变量定义在_BBS 段，对于已初始化的数据定义在_DATA 段，第二，如果所定义的常数是一个指针，则随着编译时内存模式的不同，定义格式有所不同，如表 1-4 所示。

(3)命名约定

为了与 C 语言命名约定相符，在定义和编写汇编语言程序中的被调用过程时，应以下划线开头，并用 PUBLIC 说明。此外，过程名最好不要超过 8 个字符。例如：

```
PUBLIC  _atest
_atest proc far
   ⋮
_atest endp
```

(4)参数传递约定

考虑调用约定，C 语言程序向汇编过程传送参数是通过堆栈进行的，而 C 语言参数压栈的顺序与参数在调用时参数表中出现的顺序相反(也与其他语言对参数压栈的顺序相反)。也就是说，第一个参数的地址最低，也是最后一个压入堆栈的。而且，各种类型的参数在堆栈中所占的字节数也是不同的，比如：int 型占 2 个字节，而 real 型要占 6 个字节(注意，类型对应的字节数还与机器有关，请用时小心)。C 语言的堆栈结构如图 1.11 所示。可见，如调用顺序是 test(A,B)，B 则先于 A 入栈。C 语言的参数传递方式是传值，但数组例外，数组总是采用传地址方式。

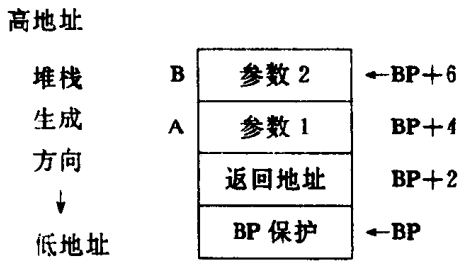


图 1.11　C 语言堆栈结构

(5)返回值约定

在被 C 语言调用的汇编过程中用指令 RET 返回，而在 BASIC 等其他语言中要求用指令 RETn 返回。因为 C 语言的调用程序会自动恢复堆栈。另外，如果 C 语言程序以 small 或者 compact 模式编译，其堆栈中的返回地址只占 2 个字节；如果 C 语言程序以 Huge，Large 或 medium 模式编译，则返回地址在堆栈中要占 4 个字节；这是在访问参数时需要注意的。

2. C 语言调用汇编程序的步骤

综上所述，我们可以列出用 C 语言调用汇编程序的步骤如下：

(1)按各种约定编写、汇编汇编语言程序得到可重定位的目标文件(obj 文件)，注意下划线，PUBLIC 和 BP 的使用。

(2)按约定编写 C 语言程序，编译后得到目标文件，注意 extern 和参数的类型的使用。

(3)将两种语言程序的目标文件连接成一个文件，得到可执行文件。

(4)执行该文件。

3. C 语言调用汇编程序的实例

例 1.11　计算 $A\times 2^B$

C 语言调用一个汇编语言程序，传送两个参数 A 和 B。汇编过程完成 $A\times 2^B$ 计算并将结果由 C 语言输出。

```
/  *C  语言程序  exam111.c  */
extern int power2 (int,int);
main( )
{
  printf("3 times 2 to the power of 5 is %d\n",power2(3,5));
}
;汇编语言程序:exam111.asm
.MODEL SMALL
.CODE
            PUBLIC  _power2
_power2     proc
            push    bp              ;保存 BP
            mov     bp,sp           ;设置堆栈指针
            mov     ax,[bp+4]       ;取参数 1
            mov     cx,[bp+6]       ;取参数 2
            shl     ax,cl           ;乘方
```

```
        pop     bp
        ret
_power2 ENDP
        END
```

这是一个极简单的程序。C 语言程序先用 extern 对被调汇编过程 power2 作了说明，变量类型为 int 型。C 语言程序用小型模式编译，在汇编程序中用了. MODEL SMALL 与之呼应，并将被调过程 power2 前加上下划线且说明成 PUBLIC，在结束时，只用指令 RET 返回。

例 1.12 捕获并显示 Ctrl-C 和 Ctrl-break

C 语言程序提示用户输入字符，并对 Ctrl-C 和 Ctrl-break 字符进行捕获，然后给于显示。汇编程序所进行的工作就是把原来 DOS 操作系统中由中断 INT 23H 和 INT 1BH 分别进行的 Ctrl-C 和 Ctrl-break 处理，用自己编写的极简单的程序块(Ctrlbrk)替换，以后每当检测到这两种控制字符就给出相应提示，否则按正常字符输出。而中断向量的替换和恢复则由两个汇编过程 charsend 和 release 完成，它们均由 C 语言所调用。C 语言程序仅完成提示功能。

```
/*   c 语言程序 exam112.c   */
#include <stdio.h>
main()
{
  int hit=0;                        /* 压键标志 */
  int c=0;                          /* 键入字符 */
  static int flag=0;
  puts("* * * * * * Please enter the character * * * * * *\n");
  puts("* * * * * * Hit (ESC) to exit program * * * * * * *\n");
  charsend(&flag); /* 调用汇编保存原中断向量和设置用户中断向量 */
  puts("* * * * * * Assembler obtained the data * * * * * * *\n");
  while((c&127)! =27)               /* 检测 ESC 字符 */
    {
      hit=kbhit();
      if(flag! =0)
        {
          puts("\n * * * * * * the Ctrl-code detected * * * * * * *\n");
          flag=0;
        }
      if(hit! =0)
        {
          puts("\n * * * * * * the normal code detected * * * * * * *\n");
          c=getch();
          putch(c);
        }
    }
    release();                      /* 调用汇编恢复原中断向量 */
    puts("* * * * * * now program ended,please try again ! \n");
}
```

```
; 汇编语言程序：exam112.asm
                name ctoa
args            equ 4
cr              equ 0dh
lf              equ 0ah
_text           segment byte public ' code'
                assume cs:_text
                public  _charsend,_release
_charsend       proc near                       ; 保存中断向量
                push bp
                mov bp,sp
                push ds
                push di
                push si
                mov ax, word ptr [bp+args]
                mov cs:flag,ax
                mov cs:flag+2,ds
                mov ax,3523h                    ; 取 int 23h 中断向量
                int 21h
                mov cs:int23,bx                 ; 保存 int 23h 中断向量
                mov cs:int23+2,es
                mov ax,351bh                    ; 取 int 1bh 中断向量
                int 21h
                mov cs:int1b,bx                 ; 保存 int 1bh 中断向量
                mov cs:int1b+2,es
                push cs
                pop ds
                mov dx,offset ctrlbrk           ;设置用户中断向量
                mov ax,2523h
                int 21h
                mov ax,251bh
                int 21h
                pop si
                pop di
                pop ds
                pop bp
                ret
_charsend       endp
_release        proc near                       ;恢复原中断向量
                push bp
                mov bp,sp
                push ds
                push di
```

```
                push si
                mov dx,cs:int1b
                mov ds,cs:int1b+2
                mov ax,251bh
                int 21h
                mov dx,cs:int23
                mov ds,cs:int23+2
                mov ax,2523h
                int 21h
                pop si
                pop di
                pop ds
                pop bp
                ret
_release        endp
_ctrlbrk        proc far                ;用户自写的中断过程
                push bx
                push ds
                mov bx,cs:flag
                mov ds,cs:flag+2
                mov word ptr ds:[bx],1
                pop ds
                pop bx
                iret
_ctrlbrk        endp
flag            dw 0,0                  ;数据暂存区
int23           dw 0,0
int1b           dw 0,0
_text           ends
                end
```

二、自动产生汇编语言框架程序

如前所述，C 语言要能实现对汇编程序正确调用，汇编程序的编写必须满足许多约定，一不小心，就会出错。下面介绍一种自动产生汇编语言框架程序的方法，会给你带来极大的方便。

假设 C 语言调用的汇编子程序函数为 Tcseg( )，则可用 Turbo C 命令行编译程序 Tcc.exe 的设置开关－s，来产生一个汇编接口框架程序，尔后在框架程序中填上完成特定功能的汇编语句。其步骤如下：

首先用 Turbo C 写一个与调用汇编程序的函数名相同的空函数，如下：

```
Tcseg( )
{ }
```

给此函数取名为 Tcseg.c 并存起来，然后使用：

Tcc -s Tcseg.c↙

进行编译，编译后自动生成一个名为 Tcseg.asm 的文件，该文件的格式如下：

```
;       tcseg.asm

        ifndef  ?? version
? debug macro
        endm
        endif
        ? debug S "tcseg.c"
_TEXT   segment byte public ' CODE'
DGROUP  group   _DATA,_BSS
        assume  cs:_TEXT,ds:DGROUP,ss:DGROUP
_TEXT   ends
_DATA   segment word public ' DATA'
d@      label   byte
d@w     label   word
_DATA   ends
_BSS    segment word public ' BSS'
b@      label   byte
b@w     label   word
        ? debug C E940B5171F0774637365672E63
_BSS    ends
_TEXT   segment byte public ' CODE'
;       ? debug L 1
_tcseg  proc    near
@1:
;       ? debug L 2
        ret
_tcseg  endp
_TEXT   ends
        ? debug C E9
_DATA   segment word public ' DATA'
s@      label   byte
_DATA   ends
_TEXT   segment byte public ' CODE'
_TEXT   ends
        public  _tcseg
        end
```

在这个框架式汇编语言程序中，只要在"@1:"处插入实现该函数特定功能的汇编语句和有关进栈、退栈及参数传递的语句，在_DATA 段填入初始化数据，在_BSS 段填入非初始化数据即可。

对于 Boriand C，汇编接口框架程序的形成方法与 Turbo C 相似，只要把 TCC 换为 BCC 即

可。若 Borland C 调用的汇编子程序函数为 bcseg( ),则自动形成的汇编语言接口框架程序 bcseg. asm 的文件格式如下。

```
;   bcsegm. asm

            ifndef      ?? version
? debug     macro
            endm
publicdll macro         name
            public      name
            endm
$ comm      macro       name,dist,size,count
            comm        dist name:BYTE:count * size
            endm
            else
$ comm      macro       name,dist,size,count
            comm        dist name[size]:BYTE:count
            endm
            endif
            ? debug     V 300h
            ? debug     S "bcsegm. c"
            ? debug     C E954B6171F0862637365676D2E63
_TEXT       segment byte public ' CODE'
_TEXT       ends
DGROUP      group       _DATA,_BSS
            assume      cs:_TEXT,ds:DGROUP
_DATA       segment word public ' DATA'
d@          label       byte
d@w         label       word
_DATA       ends
_BSS        segment word public ' BSS'
b@          label       byte
b@w         label       word
_BSS        ends
_TEXT       segment byte public ' CODE'
        ;
        ;   bcseg()
        ;
            assume      cs:_TEXT
_bcseg      proc        near
            push        bp
            mov         bp,sp
        ;
        ;{ }
```

```
;
        pop     bp
        ret
_bcseg  endp
        ? debug  C E9
        ? debug  C FA00000000
_TEXT   ends
_DATA   segment word public ' DATA'
s@      label   byte
_DATA   ends
_TEXT   segment byte public ' CODE'
_TEXT   ends
        public  _bcseg
_s@     equ     s@
        end
```

在 bcseg. asm 上编程较之在 tcseg. asm 上编程更简单，因为 bcseg. asm 已把进栈和退栈操作都设置好了，用户只需考虑怎样去实现函数的功能。

三、嵌入汇编语言

如果不想涉及在独立汇编语言模块中遇到的麻烦，Turbo C 和 Borland C 允许在 C 语言中直接编写汇编语言代码，称为嵌入汇编

1. 嵌入汇编过程的建立

要在 C 语言程序中直接建立汇编过程，必须用 asm 关键字来引入汇编指令，即 asm 必须放在每一条汇编指令之前。格式为：

asm opcode operands;or newLine

式中，opcode 是有效的 8088 指令；operands 是指令 opcode 可接收的操作数，可以引用 C 常量、变量和标号；";or newLine"是一个分号或换行，用于标记 asm 语言的结束。与汇编语言编程不同，分号不再用于表示一个注释的开始，需要注释 asm 语句时，可用 C 风格的注释形式，如 /* …… */。例如：

```
myfun( )
{
  int i;
  int x;
  if(i>0)
    asm mov x,1
  else
    x=7;
}
```

式中，汇编语句 mov x,1 完成对 C 变量 x 的赋值操作。注意，在 mov x,1 指令后没有分号。在嵌入式汇编中编译程序允许各种各样操作数，即使对汇编语言来说是错误的。

如果需要使用多个 asm 语句，可以将它们放在花括号内。例如：

```
int min(int v1,int v2)
{
  asm{
      mov ax,v1
      cmp ax,v2
      jle minexit
      mov ax,v2
      }
  minexit;
return(_AX);
}
```

式中,使用了伪变量_AX,它代表寄存器 AX。要想从 C 程序中直接访问某寄存器,可使用伪变量,其构成是在寄存器名前加一个下划线。

2. 嵌入汇编过程的编译和连接

若在 C 语言中嵌入汇编指令,其编译和连接过程如下:

(1)用命令行编译程序(TCC/BCC),指定-B 命令行可选项,来编译包含嵌入汇编指令的 C 语言程序。也可以在源文件中使用#pragma inline,它和使用-B 选项的作用一样。

(2)由第一步将生成. ASM 汇编文件,用宏汇编(V3. 0 以上的版本)或 Turbo C 汇编 TASM 将上一步生成的汇编文件编译成目标文件. OBJ。

(3)用连接程序 LINK 或 TLINK 生成可执行文件。

1.7.4 汇编语言与 BASIC 语言间的接口

目前有两种类型的 BASIC 语言,一种是解释型 BASIC 语言,另一种是编译型 BASIC 语言。这两种类型的 BASIC 语言程序与汇编语言之间的接口技术是不一样的。汇编语言与编译型 BASIC 语言的接口方式较简单,而与解释型 BASIC 语言的接口方式较复杂。考虑到其发展方向是编译型 BASIC 语言,本书仅介绍汇编语言与编译型 BASIC 语言之间的接口技术。

一、BASIC 语言调用汇编程序的方法

编译型 BASIC,如目前常用的 Quick BASIC,True BASIC,Turbo BASIC 等 BASIC 语言调用汇编过程是很方便的。在 BASIC 程序中,可以用 CALL,CALLS 或者 DECLARE 等语句建立两种语言之间的接口。自然,关键的是汇编过程的建立和参数的传递。建立一个可被编译型 BASIC 语言调用的汇编过程具有如下约定:

(1)在汇编程序中要把被调用的过程说明成远程(段间)调用过程,即

xx proc far

并将过程名说明成 public。

(2)在参数保存入栈时,其入栈的顺序与参数在 BASIC 程序中出现的顺序相一致。第一个变量位于最高存储位置,它也是第一个入栈的变量,而堆栈是向下生长。如图 1.8 所示。

(3)作为 BASIC 语言的默认形式,参数是按两个字节的位移量地址存放的,并形成传址调用,而 BASIC 语言的调用过程是一个远程(段间)调用,所以返回地址是四个字节。

(4)当汇编过程结束退出时,过程必须将寄存器SP恢复成参数入栈以前的值,这可以用RETn的返回指令类型。数值n是所有参数占有的总字节数。由于是传址调用,参数(不论是数字变量还是字符串变量)的地址都占用两个字节(对字符串变量而言是其描述字的地址),所以,可以用RET 2*m来计算,m是调用参数的总个数。

(5)按照BASIC语言的命名约定,BASIC语言输出的字符名是以大写字母表示的,这正是汇编语言的缺省默认形式。BASIC语言可以承认多达40个字符的变名标识符,而汇编语言中程序名的最大长度只允许31个字符。不过,这么长的命名范围一般不会出现命名冲突问题。

二、BASIC语言调用汇编程序的实例

例1.13 计算$A\times 2^B$

BASIC程序调用一个汇编程序过程,传送两个参数A、B到汇编过程,由汇编过程完成$A\times 2^B$的计算,然后再由BASIC语言程序打印输出结果。

```
' BASIC 语言程序:exam113.bas
'     BASIC routine calls MASM
DEFINT A-Z
PRINT  "3 times 2 to the power of 5 is";
PRINT  power2(3,5)
END
;汇编语言程序:exam113.asm
;MASM routine called by BASIC
.MODEL MEDIUM
.CODE
        PUBLIC power2
power2  proc far
        push bp
        mov bp,sp
        mov bx,[bp+8]          ;取参数1的地址
        mov ax,[bx]            ;把参数1送入ax中
        mov bx,[bp+6]          ;取参数2的地址
        mov cx,[Bx]            ;把参数2送入cx中
        shl ax,cl              ;乘方
        pop bp
        ret 4
power2  endp
        end
```

1.7.5 汇编语言与FORTRAN语言间的接口

FORTRAN语言可以采用INTERFACE语句来进行外部汇编过程的调用。然而,INTERFACE语句并不是非用不可,每当在编程者打算改变FORTRAN语言缺省形式时,INTERFACE就用来建立对应的调用接口。

在FORTRAN语言与汇编语言程序接口过程中，其语言间的约定和BASIC语言与汇编语言的约定相似。区别如下：

FORTRAN以Huge或Large模式编译，则传递参数方式是远程(段间)传址传送，参数地址为4个字节；若以medium模式编译，则是近程传址传送，参数地址为2个字节。

例1.14 计算$A\times 2^B$

用汇编完成一个$A\times 2^B$的计算，再由FORTRAN将结果输出。

```
C   exam114.FOR-FORTRAN calls MASM
    INTERFACE TO INTEGER *2 POWER2(A,B)
    INTEGER *2 A,B
    END
C
    INTEGER *2 A,B
    A=3
    B=5
    WRITE(*,*)'3 times 2 to the power of 5 is', POWER2(A,B)
    END
;汇编语言程序:exam114.ASM
;MASM routine called by FORTRAN
.MODEL LARGE
.CODE
          PUBLIC power2
POWER2    PROC
          PUSH BP
          MOV BP,SP
          LES BX,[BP+10]        ;取参数1的地址
          MOV AX,ES:[BX]        ;送参数1至AX
          LES BX,[BP+6]         ;取参数2的地址
          MOV CX,ES:[BX]        ;送参数2至CX
          SHL AX,CL             ;乘方
          POP BP
          RET 8
POWER2    ENDP
          END
```

从程序中可见，每一个参数都以4个字节长的地址压入堆栈。地址装入由LES指令完成。在退出汇编过程时，用RET 8恢复堆栈，因为这里压入堆栈的参数所占的总字节数为8，从返回地址之上算起。

第二章　系统软硬件接口技术

2.1　中断接口

2.1.1　系统中断接口方法

一、8259A 可编程中断控制器

IBM PC 采用 Intel 8259A 可编程中断控制器协助 CPU 进行中断处理，通过它可以完成：

(1)优先级排队管理。外部设备中断信号源分别接至 8259A 的 IR_0—IR_7，8259A 对它们进行优先级排队，IR_0 的优先级最高，依次降低，IR_7 最低。当同时出现两个或两个以上的中断信号时，8259A 首先响应优先权高者。

(2)接收外部设备的中断请求。经过优先级判决找到哪一个中断源的中断请求级别最高，然后再向 CPU 请求中断申请 INT。此时，CPU 是否响应外设硬件中断取决于中断允许标志位 IF 的状态。

(3)提供中断类型号。当响应中断时，8259A 把中断类型号送上双向数据总线，供 CPU 读取。CPU 通过 8259A 提供的中断类型号即可找到中断服务程序的入口地址，转移到中断服务程序去执行。8088 中断矢量表存放在存储器的最低端 1K 字节(地址为 00000－003FFH)，每 4 个字节存放一个中断服务程序的入口地址(共可存放 256 个中断服务程序的入口地址)。高地址的 2 个字节存放段地址，低地址的两个字节存放偏移地址，其值为对应的中断类型号乘 4。8259A 的 IR_0－IR_7 的中断类型号为 8－F，中断服务程序的入口地址为 0000：0020H－0000：003FH.

IBM PC 内装有一片 8259A 芯片，为 IR_0、IR_1、IR_3、IR_4、IR_5、IR_6、IR_7 均分配了标准设备，仅 IR_2 未用，用户可用于完成某种特定任务。若这些中断源仍不能满足用户的需要，用户可以多片 8259A 构成级联结构。

在 PC/AT 机中，由两片 8259A 级联为系统提供 15 级中断请求，IR_2 用于级联。IR_9 由软件重定向为 INT 0AH 以便与 PC/XT 兼容，IR_8、IR_{10}、IR_{11}、IR_{12}、IR_{15}为用户保留。

二、硬件中断的编程技术

1. 主程序结构

主程序基本结构如图 2.1，按框图讨论如下：

(1)保存原中断向量。将中断向量保存在堆栈中或自设的存储单元中。可用 DOS 系统功能调用和入栈操作实现。

```
MOV AH,35H
MOV AL,N    ;N 为原中断类型号
```

```
INT 21H
PUSH BX   ;保存偏移地址
PUSH ES   ;保存段地址
```

(2)设置用户自编中断向量。可用 DOS 系统功能调用或直接修改中断向量实现。

```
MOV AX,0
MOV ES,AX
MOV DI,N*4   ;N 为中断类型号
MOV AX,OFFSET ROUTINE   ;设置偏移地址
CLD
STOSW
MOV AX,SEG ROUTINE   ;设置段地址
STOSW
```

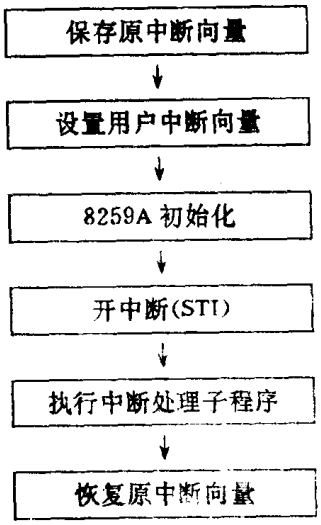


图 2.1 主程序基本结构框图

(3)8259A 的初始化。使用 8259A 时应初始化中断屏蔽寄存器(IMR),它的 I/O 口地址是 21H,位 0－位 7 对应于 IR_0－IR_7,IMR 的任意一位为 0 或 1 分别表示允许或禁止对应中断源。

```
MOV AL,0F5H   ;屏蔽 IR5,允许 IR3 中断
OUT 21H,AL
```

(4)开中断。所有中断的执行取决于程序状态寄存器的中断标志位 IF,若允许硬件中断,则应用 STI 指令将 IF 置为 1。

(5)恢复原中断向量。当中断程序结束时,必须恢复原来的中断向量,否则后续程序不能正确使用系统提供的中断系统。可用 DOS 系统功能调用和出栈操作实现。

```
POP AX   ;取出保存的段地址
POP DX   ;取出保存的偏移地址
PUSH DS   ;
MOV DS,AX
MOV AL,N   ;N 为中断类型号
MOV AH,25H   ;恢复原中断向量
INT 21H
POP DS
```

2. 中断处理子程序基本结构

图 2.2 给出了中断处理子程序基本结构框图,现分别讨论如下:

(1)保存寄存器内容。将在中断服务子程序中要修改的寄存器的内容入栈,保存现场,以便正确返回。

```
PUSH AX
PUSH BX
PUSH CX
PUSH DX
```

(2)开中断。在中断发生时,CPU 自动清除 IF 位和 TF 位,这种设计的目的是使 CPU 转入中断处理程序后,不允许再产生新的中断。如果在执行中断处理程序的过程中,还允许外部的中断,则必须通过 STI 指令把 IF 置为 1。

(3)中断服务部分。中断服务部分是中断处理程序的主体部分,它可完成各式各样的任务。如:实时控制、重大故障的紧急处理、高速数据采集控制时钟等。

(4)送中断结束命令(EOI)。在中断程序结束前,必须给 8259A 可编程中断控制器的中断命令寄存器发出中断结束命令(EOI)。中断命令寄存器的 I/O 端口地址为 20H,它的第 5 位(EOI)即为中断结束位。当 EOI 位为 1 时,当前正在处理的中断请求就被清除。所以在中断处理完成后,必须把中断结束位置 1,否则以后将屏蔽掉对同级中断或低级中断的处理,特别是键盘中断被屏蔽,会造成系统瘫痪。

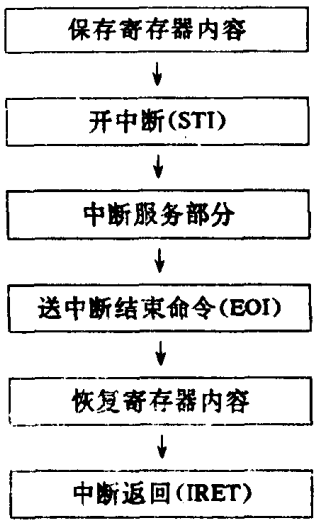


图 2.2 中断处理子程序基本结构框图

```
MOV AL,20H
OUT 20H,AL
```

(5)恢复寄存器内容。当中断结束时,,必须恢复原来的寄存器内容,否则后续程序将不能正确运行。

```
POP DX
POP CX
POP BX
POP AX
```

2.1.2 应用实例

例 2.1 系统定时中断扩展

PC 系列的定时功能是利用 INTEL 8253 可编程间隔定时器来实现的。8253 有 3 个 16 位减 1 计数器,使用频率是 1.19MHz。计数器 0 产生中断 8H,计数器 1 触发动态存储器刷新,计数器 2 驱动扬声器电路。系统上电自检初始化定时器的通道 0,OUT0 以频率 18.2Hz(1.19MHz/65536)输出一方波,直连到中断控制器 8259A 的 IRQ0 端,作为 0 级外中断请求,而中断控制器输出与 IRQ0 级中断对应的中断号 08H 到 CPU,CPU 以地址 20H(中断号×4)访问中断向量表,获得中断 8H 处理程序的入口地址。在中断 8H 中要完成更新系统日时钟及控制软盘磁头缩回等功能,并且能够提供一个每秒执行 18.2 次的软中断程序 INT 1CH。

IRQ0 已作为定时器 8253 通道 0 的中断,它每隔 55ms(每秒 18.2 次)产生一次中断,以此来作为系统日时钟。由于系统定时中断不是一个整数,时钟计算并不方便,也不很精确。因此,可以重新编写其定时中断服务程序,比如 10ms(每秒 100 次)产生一次中断。

要实现上述功能,首先要修改 8253 通道 0 的定时常数,使其 10ms 产生 IRQ0 的中断请求;同时,编写新的中断服务程序,把它的入口地址填入 0020H 单元中。程序清单如下。

```
;     exam21.asm

stack       segment stack ' stack'
            db 256 dup(0)
stack       ends
data        segment public ' data'
count100    db 100                  ;100 次一秒计数器
tenhour     db 0                    ;小时的十位数
hour        db 0                    ;小时的个位数
            db ' :'
tenmin      db 0                    ;分的十位数
minute      db 0                    ;分的个位数
            db ' :'
tensec      db 0                    ;秒的十位数
second      db 0                    ;秒的个位数
data        ends
code        segment public ' code'
start       proc far
            assume cs:code
            push ds                 ;保留 psp+0 段地址
            mov ax, 0
            push ax
            mov ax, data
            mov es, ax
            assume es:data          ;设附加段地址
            mov si,82h              ;传递参数
            mov di,offset tenhour   ;存放时间的首地址
            mov cx,8
            cld
            rep movsb               ;将八个字串从 psp 中移到存放时间的单元里
            ;建立正常数据段
            mov ds,ax
            assume ds:data          ;建立数据段地址
            mov ah,0                ;等待敲键
            int 16h                 ;启动"日时钟"
            ;建立新的定时中断服务程序
            cli                     ;禁止中断
            mov ax,0
            mov es,ax               ;指出附加段地址
            mov di,20h              ;中断类型 8 地址向量单元
            mov ax,offset timer     ;中断偏移地址->ax
            stosw                   ;存入中断向量表
            mov ax,cs               ;段地址送 ax
```

```
                stosw                       ;存入中断向量表
                ;编程计数器 0 通道
                mov al,36h                  ;8253 方式 3
                out 43h,al                  ;送控制字寄存器
                mov bx,11932                ;100hz 计数器
                mov al,bl
                out 40h,al                  ;送低字节
                mov al,bh                   ;然后送高字节
                out 40h,al
                ;编程 8259 中断控制器
                mov al,0fch                 ;允许键盘和定时器中断
                out 21h,al                  ;送中断屏蔽寄存器
                sti                         ;开放中断
                ;屏幕显示时间
forever:        mov bx,offset tenhour
                mov cx,8
dispclk:        mov al,[bx]
                call dispchar               ;屏幕显示
                inc bx                      ;取出下一个字符的地址
                loop dispclk
                mov al,0dh
                call dispchar               ;显示回车符
                mov al,second               ;取秒数
waiter:         cmp al,second               ;变了吗?
                jz waiter                   ;等待变
                jmp forever                 ;变了,显示
                ;新的定时中断服务程序
timer           proc far
                push ax                     ;保留
                dec count100                ;递减计数器的次数
                jnz timerx                  ;未到零,返回
                mov count100,100            ;到零,复位
                inc second                  ;加 1 秒
                cmp second,' 9'             ;和 9 比
                jle timerx
                mov second,' 0'             ;否则写 0
                inc tensec                  ;加 10 秒
                cmp tensec,' 6'             ;进分(同 6 比)?
                jl timerx                   ;若小于,返回
                mov tensec,' 0'             ;否则写 0
                inc minute                  ;加一分
                cmp minute,' 9'             ;进位(同 9 比)?
                jle timerx                  ;小于等于返回
```

```
            mov minute,' 0'
            inc tenmin                  ;加 10 分
            cmp tenmin,' 6'             ;进小时?
            jl timerx
            mov tenmin,' 0'
            inc hour                    ;加一小时
            cmp hour,' 9'               ;进位
            ja adjhour
            cmp hour ,' 3'              ;超过 12 小时了吗?
            jnz timerx                  ;不是返回
            cmp tenhour,' 1'            ;小时的十位为 1 吗?
            jnz timerx                  ;不是返回
            mov hour ,' 1'              ;是,置一小时
            mov tenhour,' 0'            ;0->小时十位
            jmp short timerx
adjhour:    inc tenhour                 ;加十小时
            mov hour,' 0'
timerx:     mov al,20h
            out 20h,al                  ;发 EOI 命令
            pop ax
            iret
timer       endp
            ;
dispchar    proc near
            push bx
            mov bx,0
            mov ah, 14
            int 10h                     ;调用 BIOS 显示
            pop bx
            ret
dispchar    endp
code        ends
            end start
```

2.1.3 系统中断接口实验

实验 2.1 自定义软件中断实验

自己定义一个软件中断,中断类型码为 79H。

实验要求:

1. 在中断服务程序中,完成 ASCII 码到 BCD 码的转换,ASCII 码内存首地址为 ASCMM,字节长度为 NUMB,转换后的 BCD 码放在以 BCDM 为首地址的存储区中。

2. 能正确地转到中断服务子程序,即需设置中断向量。

3. 编写中断服务子程序和主程序。

2.2 定时与声音接口

2.2.1 定时与声音接口方法

一、8253/8254 定时器/计数器

PC 系列微机均采用 8253/8254 可编程定时/计数芯片。8253/8254 有三个通道，其端口地址为 40H～42H，控制端口地址为 43H。基本任务是：(1)通道 0 为系统日历时钟所有，系统启动时由 BIOS 置数，每秒约发出 18.2 个脉冲，定时提出日时钟中断请求，从而引发中断 08H，执行一次中断 08H 服务程序完成三件事，即日历时钟计时、控制软驱马达关闭时间和调用定时报时中断 1CH。通道 0 的输出也用于控制软驱马达关闭时间，因此，如果改变通道 0 的计数值，必须保证在 CPU 每次访问磁盘前恢复原有读数，系统初始化中断 1CH 的向量仅指向一条 IRET 指令，故而，若用户未在应用程序中接管该向量，那么调用中断 1CH 无任何意义。通常由用户接管 INT 1CH，使之指向自行编制的中断服务程序，以完成对各类数据的采集及对事件的实时处理。(2)通道 1 用于动态 RAM 刷新。(3)通道 2 产生单一方波信号用于发声。与其他通道相比，程序员对通道 2 有更多的控制权。

二、定时接口方法

由前述可知，系统日时钟计时频率是选用最大的除数(65536)得出的，每秒仅 18.2 次，往往不能满足某些实际的需求。例如，在应用中要求将计时频率提高 8 倍，但在不改变任何硬件的情况下，要不影响日时钟的准确性以及原有定时中断所完成的其他两件事：控制软驱马达关闭时间和调用报时中断 1CH。这样，就需要重新编写新的日时钟定时中断程序。其思路可用下述步骤描述。

1. 执行一个初始化流程

该流程主要完成以下几件事：

(1)接管中断 08H 向量，使之指向一个新的日时钟定时中断程序。当然，在接管前要获取原来的中断 08H 向量并保存在代码段可寻址的变量中；

(2)重新初始化定时器，使通道 0 计数器每秒中断 18.2×8 次，即加快频率，满足其要求；

(3)若应用程序不驻留内存，初始化流程结束即刻开放中断，进入主程序流程。否则，开放中断后使程序驻留，初始化便告退出。

2. 主程序按应用要求设计

主程序按 8 倍的原频率工作，并在运行结束前恢复原中断 08H 向量，否则，可能导致系统混乱。

3. 新的定时中断服务程序

三、声音接口方法

为了具有音响输出的能力，系统板上装有一个 $2\frac{1}{4}$ 英寸的扬声器以及控制电路和驱动

电路。控制电路能以位触发和定时器控制两种不同的方式驱动扬声器发声。

(1)位触发方式

程序直接控制 PPI(8255A 可编程序外围接口芯片)的输出控制寄存器(I/O 端口为 61H)的第 1 位,使该位按所需的频率进行 1 和 0 的交替变化,从而控制开关电路产生一串脉冲波形,这些脉冲经放大后驱动扬声器发出声音。如果控制这一串脉冲波形的脉宽和长度就可以产生不同频率和不同音长的声音。

采用位触发方式发声的程序段如下:

```
        IN AL,61H
        MOV AH,AL
        AND AL,11111100B        ;关断定时器通道 2 的门控
SOUND:  XOR AL,2                ;触发 61H 端口第 1 位
        OUT 61H,AL
        MOV CX,DX               ;(DX)=控制脉宽的计数值
WAITER: LOOP WAITER             ;延时循环
        DEC BX                  ;(BX)=脉冲持续的时间
        JNZ SOUND
        MOV AL,AH
        OUT 61H,AL              ;恢复 61H 端口
```

(2)利用定时器产生声音

这是利用机器硬件即 INTEL 8253/8254 定时器产生声音的一种方法。

CPU 通过对定时器的通道 2(端口地址为 42H)进行编程,使其 I/O 寄存器接收一个控制声音频率的 16 位计数值,端口 61H 的最低位控制通道 2 门控的开断,以产生特殊的音响。当定时器接收的计数值为 533H 时,能产生 896Hz 的声音,因此产生其他频(Freq)的计数值就可由下式计算出来:

1331×896÷给定频率=1193180(1234DCH)÷给定频率

若给定频率在 DI 中,可用下述程序产生相应的常数(送 8253—5 通道 2):

```
MOV DX,12H
MOV AX,34DCH
DIV DI
```

发声的持续时间以 10ms 为单位,10ms 的延时(持续)时间可以用 DL10ms 来实现,即:

```
        MOV CX,2801
DL10ms: LOOP DL10ms
```

这是因为 MOV 指令为 4 个 T,LOOP 指令为 17T(产生转移)或 5T(不产生转移),每个 T 为 210ns,故:

$$[17(n-1)+5+4]\times 210\times 10^{-9}=0.01$$

$$n=2801$$

对于 IBM-PC/AT 机,MOV 指令为 2 个 T,LOOP 指令分别为 8T/4T。对于 10MHz(晶振为 20MHz)的 80286 微处理器,每个 T 为 100ns,则:

$$[8(n-1)+4+2]\times 100\times 10^{-9}=0.01$$

$$n=12500.25\doteq 12500$$

2.2.2 应用实例

例 2.2 乐曲"两只老虎"

编制一个演奏特定乐曲的程序，通常按下述步骤进行：

1. 设置特定乐曲的频率表和持续时间表；
2. 将两张表的偏移地址分别置于 SI 和 BP；
3. 调用通用发声程序 SING 演奏乐曲。

乐曲《两只老虎》的简谱如下：

两只老虎

1=C 4/4

1 2 3 1 | 1 2 3 1 | 3 4 5 — |

3 4 5 — | 5̲6̲ 5̲4̲ 3 1 | 5̲6̲ 5̲4̲ 3 1 |

2 5 1 — | 2 5 1 — |

数据段中定义了频率数据(FREQ)和节拍时间数据(TIME)，0000H 作为频率数据结束标志。代码段中，演奏过程(SING)要求把频率数据的首地址送到 SI，节拍时间数据的首地址送到 BP。

```
;       exam22.asm

stack   segment para stack ' stack'
        dw   100 dup(?)
stack   ends
data    segment
bg      db 0ah,0dh,"two tiger; $"
freq    dw 2 dup(262,294,330,262)              ;频率数据
        dw 2 dup(330,349,392)
        dw 2 dup(392,440,392,349,330,262)
        dw 2 dup(294,196,262),0
time    dw 10 dup(25),50,25,25,50              ;时间数据
        dw 2 dup(12,12,12,12,25,25)
        dw 2 dup(25,25,50)
data    ends
code    segment
        assume cs:code,ds:data
stat    proc far
        push ds
        mov ax,0
        push ax
        mov ax,data
        mov ds,ax
        mov dx,offset bg                       ;显示歌名
        mov ah,09
```

```
            int 21h
            mov si,offset freq
            mov bp,offset time
            call sing                    ;调用 SING 过程
            ret
stat        endp
sing        proc near
            push di
            push si
            push bp
            push bx
repter:     mov di,[si]
            cmp di,0
            je end_sing
            mov bx,ds:[bp]
            call sound
            add si,2
            add bp,2
            jmp repter
end_sing:   pop bx
            pop bp
            pop si
            pop di
            ret
sing        endp
sound       proc near                    ;声音过程
            push ax
            push bx
            push cx
            push dx
            push di
            mov al,0b6h                  ;8253 初始化(选通道 2,产生方波信号)
            out 43h,al                   ;43 端口是 8253 的命令寄存器
            mov dx,12h                   ;计算时间常数
            mov ax,34deh
            div di
            out 42h,al                   ;设置时间常数
            mov al,ah
            out 42h,al
            in al,61h
            mov ah,al
            or al, 3
            out 61h,al                   ;开喇叭(8255 I/O 端口 61h 的低两位置 1)
```

```
delay:    mov cx,12500          ;延时
d110ms:   loop d110ms
          dec bx
          jnz delay
          mov al,ah
          out 61h,al            ;关喇叭
          pop di
          pop dx
          pop cx
          pop bx
          pop ax
          ret
sound     endp
code      ends
          end stat
```

2.2.3 声音接口实验

实验 2.2 乐曲"两只老虎"

实验要求：用位触发方式编写程序，使 PC 机发出音乐并奏出"两只老虎"的乐曲。

2.3 键盘接口

2.3.1 键盘接口方法

一、8255A－5 和 8048 芯片的使用

在 PC 机中，对键盘的管理是通过 8255 和 8048 芯片来实现的。

在键盘内部，有一个微处理器 INTEL 8048，该处理器从系统板接收时钟信号，并读取每个键入的字符，将其扫描码放在 8255 外围接口芯片的 PA 端口(60H)内。PB 端口(61H)的第 6 位控制着键盘的时钟信号，当键盘正常工作时，第 6 位应总是 1，否则将锁闭键盘。PB 端口的第 7 位置 1 时，可发送一个应答信号给键盘微处理器。每当按下键或放开键，在 8048 将其扫描码(通码或断码)送入 PA 端口的同时，还产生一个类型为 09H 的中断，该中断的任务是：1. 读扫描码并把应答信号送到键盘；2. 把扫描码转换成字符码或变换键状态；3. 在键盘缓冲区内设置键的字符码。用户自编键盘中断程序，可重新定义键盘上的任何键。

二、键盘接口方法

键盘接口有两种方法：

1. 直接在硬件基础上编程。从 8255PA 口读取键盘扫描码，从 PB 口读状态信息或发送应答信号：

```
IN AL,60H        ;从 PA 口读扫描码
```

```
PUSH AX
IN AL,61H        ;读 PB 口信息
OR AL,80H
OUT 61H,AL       ;置键盘应答位
AND AL,7FH
OUT 61H,AL       ;复位键盘应答位
```

2. 键盘的 DOS 和 BIOS 功能调用

BIOS 调用采用软中断 INT 16H 实现，其中有三种功能：功能号 0(从输入缓冲区读一个字符，若空则等待)，功能号 1(从输入缓冲区读一个字符，不管缓冲区是否为空，均返回)，功能号 2(读取键盘状态)。键盘的 DOS 调用，其功能比 BIOS 调用丰富得多，主要有：功能号 1(键盘输入)、功能号 6(直接控制台输入/输出)、功能号 7(直接控制台输入但不输出/显示)、功能号 8(控制台输入但不显示)和功能号 A(字符串输入)、功能号 B(检查键盘输入状态)。利用键盘的 DOS 和 BIOS 功能调用比直接在硬件基础上编程要方便和简单得多。

2.3.2 应用实例

例 2.3 键盘处理软件

键盘是通过中断方式工作的，考虑到键盘产生中断和在计算机中运行的主程序的响应是异步的，即有键可能随时被按下，而主程序何时接收键盘输入却不是随时都能进行的，即当时进行的工作不能暂停，因此键盘处理程序应用一缓冲区，一旦有键输入时，它就保存这个输入，直到程序取走它为止。程序中缓冲区定义在数据段，并取名为 BUFFER，它能存下按入的十个键的 ASCII 码，名为 BUFPTR1 和 BUFPTR2 的两个指针用来指示在缓冲区中开始和结束的数据，当 BUFPTR1＝BUFPTR2 时，说明缓冲区中没有数据，程序循环等待下一次键盘中断输入。为指针增加 1 后超出缓冲区(BUFFER＋10)时，头尾指针又重回到缓冲区的开始(BUFFER＋0)。如果缓冲区存满时接收了字符，则就简单地忽略这个字符。

```
;     exam23.asm

stack     segment   para stack ' stack'
          db          256 dup(0)
stack     ends
data      segment   para public ' data'
buffer    db   10 dup(0)                    ;键盘缓冲区 BUFFER
bufptr1   dw   0                            ;BUFFER 的开始指针
bufptr2   dw   0                            ;BUFFER 的结束指针
          ;现在 bufptr1＝bufptr2,即 buffer 是空的
scantable db   0,0,' 1234567890－＝' ,8,0
          db   ' qwertyuiop[ ]' ,0dh,0
          db   ' asdfghjkl;' ,0,0,0,0
          db   ' zxcvbnm,./' ,0,0,0
          db   '  ' ,0,0,0,0,0,0,0,0,0,0,0,0,0
          db   ' 789－456＋1230.'
data      ends
```

```
code    segment  para public ' code'
start   proc     far
        ;标准的程序框架
        assume   cs;code
        push     ds
        mov ax,0
        push ax
        mov ax,data
        mov ds,ax
        assume ds;data
        ;第一部分:设置自己的键盘中断服务程序
        cli                                  ;禁止所有的中断
        mov ax,0
        mov es,ax                            ;指向附加段(同数据段地址)
        ;中断服务子程序地址表
        mov di,24h                           ;类型码 09H 的偏移为 24H
        mov ax,offset kbint                  ;键盘中断程序 KBINT 偏移-->AX
        cld                                  ;地址增加方向存储
        stosw
        mov ax,cs                            ;KBINT 的段地址->AX
        stosw                                ;装入中断向量表中
        mov al,0fch                          ;允许时钟和键盘中断
        out 21h,al                           ;写到中断屏蔽寄存器
        sti                                  ;开中断
        ;第二部分:读键盘并显示字符
forever: call kbget                          ;等待键盘输入字符
        push ax
        call dispchar                        ;显示接收的字符
        pop ax
        cmp al,0dh                           ;是回车符吗?
        jnz forever                          ;否,则转,是,加换行
        mov al,0ah
        call dispchar                        ;若是,则显示
        jmp forever                          ;循环
        ;调用 KBGET,等待从键盘接收字符
kbget   proc  near
        push bx
        cli                                  ;禁止中断
        mov bx,bufptr1                       ;开始指针
        cmp bx,bufptr2                       ;缓冲区空否
        jnz kbget2                           ;否,取字符
        sti                                  ;是,开中断
        pop bx
```

```
        jmp kbget                       ;循环等待输入字符
kbget2: mov al,[buffer+bx]              ;取字符
        inc bx                          ;开始指针加一
        cmp bx,10                       ;是否到尾
        je  kbget3                      ;否,转
        mov bx,0                        ;是,指针指向始端
kbget3: mov bufptr1,bx
        sti                             ;开中断
        pop bx
        ret                             ;返回
kbget   endp
        ;用户的键盘中断服务程序
kbint   proc far
        push bx                         ;保存将要修改的各寄存器
        push ax
        ;读键盘中断扫描码,并发应答信号
        in al,60h                       ;读键盘输入
        push ax                         ;保存它
        in al,61h                       ;8255 PB 口
        or al,80h                       ;设置键盘应答信号
        out 61h,al                      ;发送键盘应答信号
        and al,7fh
        out 61h,al                      ;恢复 8255 原 PB 口,清应答信号
        ;转换扫描码为 ASCII 字符
        pop ax
        test al,80h                     ;键是否松开
        jnz kbint2                      ;是,忽略并转移
        mov bx,offset scantable         ;取表指针
        xlatb                           ;扫描码转换为 ASCII 码
        cmp al,0                        ;是有效键符吗?
        jz kbint2                       ;否,忽略它
        ;将转换后的 ASCII 码送到缓冲区去
        mov bx,bufptr2                  ;取尾指针
        mov [buffer+bx],al              ;字符送缓冲区
        inc bx                          ;尾指针增量
        cmp bx,10                       ;是否到尾
        je kbint2                       ;否,转
        mov bx,0                        ;是,指向始端
kbint3: cmp bx,bufptr1                  ;缓冲区满否?
        jz  kbint2                      ;满,丢掉字符
        mov bufptr2,bx                  ;尾指针指向新的
        ;中断结束,并返回
kbint2: mov al,20h                      ;中断结束命令
```

```
            out 20h,al
            pop ax
            pop bx
            iret
kbint       endp
            ;显示字符子程序
dispchar    proc near
            push bx                    ;保存 BX
            mov bx,0                   ;显示 0 页
            mov ah,14                  ;写功能调用
            int 10h                    ;BIOS 显示子程序调用
            pop bx                     ;返回调用该子程序处
            ret
dispchar    endp
start       endp
code        ends
            end start
```

2.3.3 键盘接口实验

实验 2.3 扩充键盘处理功能

实验要求：在示例 2.3 程序的基础上，增加如下功能：

1. 保存和恢复原中断向量；
2. 按“END”键结束键盘输入，返回 DOS；
3. 增加“Caps Lock”键的功能，即实现大小写字母转换。

2.4 显示器接口

2.4.1 显示器接口方法

一、6845 CRT 控制器

IBM 微型机的视频系统都是以 6845 CRT 控制器或以基于 6845 的定制芯片为核心构成的。6845 CRT 控制器的基本任务是：(1)将 CRT 工作方式设置成字符或图形方式之一；(2)对 ASCII 代码号进行译码，并从适配器 ROM(有时从 RAM)芯片中取出对应字符的数据；(3)对数据进行解码，以得出属性或彩色并依此调整屏幕显示；(4)生成并控制光标。

二、显示器接口方法

显示器接口有两种方法：(1)存储器映射法；(2)BIOS 显示功能调用。两种方法各有优缺点：BIOS 显示功能调用编程量小，保持了各种 PC 机之间的兼容性，但显示速度相对较慢；存储器映射法编程可大大地提高显示速度，但编程量较大，兼容性差，本书仅以 CGA 为代表。

1. 存储器映射法

通过编程将字符数据或象素值直接写入显示存储区(VRAM),这种方法称为"存储器映射"。该方法的关键在于计算屏幕显示位置与显示存储区的偏移地址之间的关系。显示存储区的段地址与显示适配器的类型有关,单色适配器为 B000H,彩色图形适配器为 B800H。

在文本方式下,屏幕上的字符位置所对应的显示存储区的偏移地址为:

行号×160+列号×2

在 320×200 图形方式下,屏幕上的一个象素所对应的显示存储区的偏移地址为:

主偏移地址+行号/2×80+列号/4

偶数行的主偏移地址为 0,奇数行的主偏移地址为 2000H。

2. BIOS 显示功能调用

显示器的 BIOS 调用采用 INT 10H 来实现:(1)功能号 0 调用设置显示方式;(2)功能号 9(写有属性字符),功能号 0AH(写无属性字符)和功能号 0EH(写字符类似打印机方式)调用实现字符显示接口;(3)功能号 0BH(确定色彩)、功能号 0CH(画点)和功能号 0DH(读点)调用实现图形显示接口。

2.4.2 显示器接口实例

例 2.4 一张移动的"笑脸"

"笑脸"字符的 ASCII 码为 02H,要使笑脸动起来,可按如下步骤:

1. 在屏幕上显示笑脸。
2. 延迟一定时间周期,这样使图形更清晰。
3. 清除笑脸(可用清除部分屏幕或用空字符在原位置重画一次来实现)。
4. 改变笑脸的行,列坐标。
5. 返回第一步,重复上述过程。

```
;     exam24.asm

stack       segment para stack ' stack'
            dw   100 dup(?)
stack       ends
code        segment para public ' code'
move_face   proc far
            assume cs:code,ss:stack
start:      push ds
            sub ax,ax
            push ax
            mov ah,15                    ;置 AH 为显示页
            int 10h
            mov ah,0
            mov al,2                     ;选择 80*25 黑白显示方式
            int 10h
            mov cx,1                     ;字符计数为 1
            mov dx,0                     ;从(0,0)开始
            sti                          ;允许中断
```

```
set_crsr:   mov ah,2                        ;移动光标到下一位置
            int 10h
            mov al,2                        ;显示笑脸
            mov ah,10
            int 10h
            call delay                      ;等待半秒钟
            sub al,al                       ;擦掉笑脸
            mov ah,10
            int 10h
            inc dh                          ;移向下一行,下一列
            inc dl
            cmp dh,25                       ;最后一行否
            jne set_crsr
            ret
move_face   endp
            ;用 INT 1AH 的功能 0 延迟 0.5 秒
delay       proc
            push bx
            push cx
            push dx
            mov ah,0
            int 1ah
            add dx,9                        ;加上延迟值(0.5/0.055)
            mov bx,dx
            ;.....不断检测 BIOS 的日历计数
repeat:     int 1ah                         ;再取日历计数值
            cmp dx,bx                       ;与延迟值比较
            jne repeat
            pop dx
            pop cx
            pop bx
            ret
delay       endp
code        ends
            end start
```

例 2.5 画等腰三角形

采用中分辨率图形显示方式(320×200),在屏幕上显示出一个红边等腰三角形,背景色为白色。该程序中有一个画点子程序 DRAW,当该点的行号和列号给出后,便调用 DRAW 子程序,将红色的象点代码写回到原所存字节的相应两位中,并重写回 VRAM 的原地址去,这样,对应行、列显示位置的象点马上改变为红点,如此调用 21 次,改写相应象点代码的两位值为红色,便可显示出一个由 21 个红点围成的等腰三角形。由此可以看出,所谓画图实际为改变屏幕上某些象点颜色,从而构成一幅彩色画面。

```
;       exam25.asm

stack       segment para stack ' stack'
            db  256 dup(0)
stack       ends
data        segment  para public ' data'
color       db       10101010b          ;每2位为一象点代码,表示为红色
masks       db       11000000b          ;下面4个分别为每一象点代码的
            db       00110000b          ;屏蔽码
            db       00001100b
            db       00000011b
count       dw       24                 ;要画的三角形的象点数
            ;下面是要画三角形各象点的坐标,字节表示为各象点的行号,字为列号
            ;共24个点
coordinates db       103                ;点的行号
            dw       154                ;点的列号
            db       103
            dw       155
            db       103
            dw       156
            db       103
            dw       157
            db       103
            dw       158
            db       103
            dw       159
            db       103
            dw       160
            db       103
            dw       161
            db       103
            dw       162
            db       103
            dw       163
            db       103
            dw       164
            db       103
            dw       165
            db       103
            dw       166
            db       102
            dw       155
            db       102
```

```
        dw      165
        db      101
        dw      156
        db      101
        dw      164
        db      100
        dw      157
        db      100
        dw      163
        db      99
        dw      158
        db      99
        dw      162
        db      98
        dw      159
        db      98
        dw      161
        db      97
        dw      160
eighty  db      80              ;表示一行的字节数
data    ends
code    segment para public ' code'
start   proc  far
        ;标准的程序框架开始部分
        assume cs:code
        push   ds               ;保存 PSP 段地址
        mov   ax,0
        push ax                 ;保存返回的偏移地址
        mov ax,data
        mov ds,ax               ;建立数据段地址
        assume ds:data
        ;第一部分:对彩色适配器初始化
        mov ah,0
        mov al,4                ;320 * 200 中分辨显示方式
        int 10h
        mov dx,3d9h             ;颜色寄存器口地址->DX
        mov al,0fh              ;选白色背景,绿红黄色组
        out dx,al               ;将设置送入颜色寄存器
        ;对彩色适配器初始化,使 VRAM 各单元为 0,当由 BIOS 调用时显示
        ;背景色
        mov ax,0b800h           ;VRAM 的段地址
        mov es,ax               ;ES 为 VRAM 的段地址寄存器
        ;第二部分:调用画点子程序,以画出 24 个点的等腰三角形
```

```
        mov cx,count                ;坐标点计数值->CX
        mov bx,offset coordinates   ;坐标表偏移地址->BX
main:   mov al,[bx]                 ;AL 中为行坐标
        inc bx
        mov dx,[bx]                 ;DX 中为列坐标
        add bx,2                    ;指向下一个坐标
        call draw                   ;调画点子程序
        loop main                   ;继续画,直到要求的点画完
        mov bx,1000                 ;延时
delay:  mov cx,2801
delay10: loop delay10
        dec bx
        jnz delay
        mov ah,0
        mov al,2                    ;80*25 黑白显示方式
        int 10h
        ret
start   endp
        ;下面是画点子程序,该点的行号在 AL 中,列号在 DX 中,象点的代码
        ;在 COLOR 变量中
        ;要修改的寄存器有:AX,DX,DI,SI
draw    proc near
        ;首先检查是偶数行,还是奇数行,从而可知主偏移是 0,还是 2000H
        shr al,1                    ;行地址字右移一位,由进位位可知是偶行
        ;还是奇行
        ;行地址字最后 1 位在进位位中,剩下的位乘 80 便得辅助偏移地址
        jc odd                      ;若是奇行则转移
        ;若偶行则执行下面内容
        mov di,0                    ;DI 为主偏移地址
        jmp short common
odd:    mov di,2000h                ;奇行主偏移地址在 DI 中
        ;将 AL 中数乘 80 便得辅助偏移地址
common: mul eighty                  ;AX 是辅助偏移地址
        add di,ax                   ;DI 为行偏移地址
        ;这样就得到了点的行字节的地址,还要知道点所在字节
        mov si,dx                   ;保存 DX 中的列号
        shr dx,1
        shr dx,1                    ;右移 2 次
        add di,dx                   ;得该字节偏移地址
        and si,03h                  ;取点在字节中的位置
        ;下面将根据修改点在字节中的位置,将其两位象点代码
        ;修改为红色,其他位仍为白背景色,然后将该字节重写回
        ;VRAM 的原地址去,从而在屏上显示出该象点。
```

```
                mov al,[masks+si]          ;得到点的屏蔽码
                mov dh,color               ;将4个象点代码->DH
                and dh,al                  ;得到点的代码,其他点的代码为0
                not al                     ;将点的屏蔽码取反
                mov ah,es:[di]             ;得到当前字节
                and ah,al                  ;将修改点代码置0
                or ah,dh                   ;使修改点代码表示红色
                ;其他点也保持不变
                mov es:[di],ah             ;将4个点代码送回VRAM中
                ret                        ;返回调用处
draw            endp
code            ends
                end   start
```

2.4.3 显示器接口实验

实验 2.4 菜单设计实验

实验要求：

1. 清屏幕

2. 开一窗口，左上角坐标为(14H,06)，右下角坐标为(40H,11H)。

3. 在窗口内，显示：

MAIN MENU

(1)Edit

(2)Save

(3)Print

(4)Quit

实验 2.5 “骑马”飞奔的实验

用画点的方式画一个骑士驱马飞奔，使马边跑边叫。

实验要求：

1. 对显示存储器 VRAM 直接存取来画一匹马和骑在马背上的骑士。即计算出马和骑士各点的坐标位置，建立一张表。

2. 编写显示马和骑士的程序，使它们动起来。

3. 若有可能的话，同时使用声音接口使马会叫。

2.5 磁盘接口

2.5.1 磁盘接口方法

实现磁盘接口有两种方法：一是直接对磁盘控制器编程，二是 DOS 系统调用和 BIOS 调用。对系统或应用程序而言，若没有特殊应用要求，一般不必直接对磁盘控制器编程。因为 DOS 系统调用和 BIOS 调用所提供的诸多功能对大多数场合是足够的。

一、磁盘的 DOS 调用

DOS 调用实现磁盘 I/O 有两种方法：一是文件控制块方法，二是文件代号方法。前者不能访问当前目录以外的文件，作随机记录 I/O 非常不方便。后者可访问当前目录以外的文件，且使得随机记录访问非常方便。

1. 使用文件控制块的方法

使用文件控制块(FCB)访问文件可采用如下步骤。

(1)在数据段中用 DB 伪指令定义文件控制块。

(2)初始化文件控制块，在 FCB 中相应字节填入盘号、文件名和后缀。盘号、文件名和后缀可作为执行程序的参数同程序名一起键入，如：A>exam example. dat↙，这样 DOS 在装入程序时将文件名填入 PSP 的 FCB 中(DS:5C～DS:67)。也可在数据段中用伪指令 DB 定义。如

```
myFCB   DB   0              ;驱动器缺省
        DB   'EXAMPLE'      ;文件名
        DB   'DAT'          ;后缀
        DB   25DUP(0)       ;FCB 剩余填 0
```

(3)打开文件(功能 0FH)或建立文件(功能号 16H)

(4)如不使用默认记录大小，应在 FCB 中填入记录长度。

(5)如执行随机记录 I/O，在 FCB 中填入随机记录号。

(6)使用功能 1AH 建立数据传送区 DTA。

(7)执行读、写记录操作(功能号 14H—顺序读，15H—顺序写，21H—随机读，22H—随机写，27H—随机块读，28H—随机块写)。

(8)关闭文件(功能号 10H)

2. 使用文件代号的方法

使用文件代号访问文件可采用如下步骤：

(1)定义文件名

文件名用所谓的 ASCII Z 串表示，ASCII Z 串指文件路径名的 ASCII 字符串后面再人为的加上一个 0 字节，如要建立的文件名为：C:\TC\LIB，ASCII Z 串则表示为'C:\TC\LIB','0'。文件名可在数据段用伪指令 DB 定义，也可用带缓冲的输入服务功能(INT 21H 功能号为 0AH)。

(2)使用 INT 21H 功能号 3DH 打开文件，或使用功能号 3CH 生成文件，并保存返回的文件代号。

(3)若执行随机记录 I/O，使用 INT 21H 功能号 42H 建立移动文件指针。

(4)读文件(功能号 3FH)或写文件(功能号 40H)。

(5)关闭文件(功能号 3EH)。

二、磁盘的 BIOS 功能调用

DOS 为用户提供的仅是磁盘文件 I/O 功能，但是在实际的应用中，有很多时候不是以文件为处理背景，而是以磁盘的物理空间为处理背景。例如，复制一个磁盘，读取磁盘中某一磁

道某扇区中的内容，这时只有借助于磁盘 BIOS 调用来实现。ROM－BIOS 支持的磁盘 I/O 程序(INT 13H)为用户提供了磁盘复位、获取磁盘操作状态、扇区读/写/校验/格式化等功能。

2.5.2 应用实例

例 2.6 用 FCB 方法分页显示文件

该程序的作用是分屏显示文本文件内容。若该程序名为 exam26.exe，是可用如下命令来分屏显示文本文件 exam26.dat 的内容。

C>exam26 exam26.dat↙

```
;     exam26.asm

stack       segment  para stack ' stack'
            db 256 dup(0)
stack       ends
data        segment para public ' data'
fcb         db  36 dup(0)                ;文件控制块
linecount   db  0                        ;屏幕上显示的计数
charpos     db  0                        ;字符在行中的位置
dta         db  0                        ;盘数据传输区
errmsg      db  ' file access error!!! '
data        ends
code        segment para public ' code'
start       proc  far
            ;标准程序框架
            ;
            assume cs:code
            push ds
            mov ax,0
            push  ax
            mov ax,data
            mov es,ax
            assume es:data
            ;将 FCB 参数从 PSP 移到数据段内
            mov si,5ch                   ;源串在 PSP 中的偏移
            mov di,offset fcb            ;目的串偏移
            mov cx,12                    ;移动的串长度
            cld                          ;设置方向是地址增加的
            rep movsb                    ;把参数移到数据区内
            ;建立对数据段的可寻址性
            mov ds,ax
            assume ds:data
            ;设置 DTA 和打开文件
            mov dx,offset dta            ;DTA 地址偏移
```

```
        mov ah,1ah
        int 21h                         ;设置 DTA 的 DOS 功能调用
        mov dx,offset fcb               ;FCB 偏移
        mov ah,0fh
        int 21h                         ;打开文件 DOS 功能调用
        cmp al,0                        ;打开文件对否?
        jnz error                       ;错! 转错误信息显示
        ;在 FCB 中建立记录大小,当前块号,当前记录号
        mov word ptr fcb+0ch,0          ;当前块号为 0
        mov word ptr fcb+0eh,1          ;记录为一个字节
        mov fcb+20h,0                   ;当前记录为 0
        ;从文件中读一个字符,并将其显示在屏幕上,并检查控制
        ;字符"TAB"(09H),文件结束(1AH)和换行(0AH)并作出相应的处理.
again:  mov dx,offset fcb
        mov ah,14h
        int 21h                         ;DOS 顺序读功能调用
        cmp al,0                        ;读正确?
        jnz error                       ;否,转错误显示
        mov al,dta                      ;在 AL 中得到字符
        cmp al,1ah                      ;是 Ctrl Z?
        jz  eof                         ;转移到 EOF
        cmp al,09h
        jz  tab                         ;是 TAB 符转移到 TAB
        call dispchar                   ;显示得到的字符
        inc charpos                     ;字符位置加 1
        cmp dta,0ah
        jnz again                       ;若不是文件结束,继续上述过程
        mov charpos,0                   ;恢复字符位置的初值
        inc linecount                   ;行计数加 1
        cmp linecount,24                ;屏是否填满
        jnz again                       ;否! 继续上述过程
        ;当屏满时,等待按键
        mov ah,0
        int 16h                         ;BIOS 键盘调用
        ;当按键后,再显示下一屏
        mov linecount,0                 ;重新使行计数为 0
        jmp again
        ;当是 TAB 字符时,便显示空格,直到遇到停止标志为止
tab:    mov al,' '
        call dispchar                   ;显示一个空格
        inc charpos                     ;字符位置加 1
        test charpos,7                  ;是停止标志
        jz  again                       ;如果是转移
```

```
            jmp   tab                          ;不是,继续显示空格
            ;文件结束,关闭文件并返回 DOS
eof:        mov dx,offset fcb
            mov ah,10h
            int 21h                            ;DOS 关闭文件功能调用
            ret                                ;返回 DOS
            ;当文件存取出错时,显示错误信息并退出
error:      mov bx,offset errmsg
            call display                       ;显示错误信息
            ret                                ;返回 DOS
            ;在屏上显示一行字符子程序,显示字符的地址在 BX 中,假设要显示的一行
            ;字符有 30 个字符
display     proc near
            mov cx,30                          ;要显示的字符数
disp1:      mov al,[bx]                        ;得到要显示的下一个字符
            call dispchar
            inc bx
            loop disp1                         ;循环 30 次
            mov al,0dh                         ;回车
            call dispchar
            mov al,0ah                         ;换行
            call dispchar
            ret                                ;返回到调用处
display     endp
            ;在屏幕上显示一个字符子程序,AL 中是要显示的字符,使用 BIOS 显示调用
dispchar    proc near
            push bx                            ;保存 BX
            mov bx,0                           ;选择 0 页显示
            mov ah,14
            int 10h                            ;在屏上写功能的 BIOS 调用
            pop bx
            ret
dispchar    endp
start       endp
code        ends
            end start
```

例 2.7　用文件代号方法复制文件

利用文件代号读写文件需要一个 ASCII 码字符串指示文件名,这个问题通过用户键盘输入的方法解决。下述程序可以将一个指定的已存在的文件拷贝成另一个指定的文件。

```
;     exam27.asm

stack       segment   para stack ' stack'
            db   100 dup(0)
```

```
stack    ends
data     segment
sfile    db 64
         db ?
         db 64 dup(' ')
dfile    db 64
         db ?
         db 64 dup(' ')
ask1     db 0ah,0dh,' please input source '
         db ' file name——>' ,' $'
ask2     db 0ah,0dh,' please input destnation '
         db ' file name——>' ,' $'
note     db 0ah,0dh,' please insert diskettes '
         db ' and strike any key when ready' ,' $'
er1      db 0ah,0dh,' create error' ,' $'
er2      db 0ah,0dh,' open error' ,' $'
er3      db 0ah,0dh,' read error' ,' $'
er4      db 0ah,0dh,' write error' ,' $'
er5      db 0ah,0dh,' close source file error' ,' $'
er6      db 0ah,0dh,' close dest file error' ,' $'
bufr     dw ?                              ;存放文件标记
data     ends
code     segment
         assume cs:code,ds:data,es:data
start    proc far
         push ds
         sub ax,ax
         push ax
         mov ax,data
         mov ds,ax
         mov es,ax
         lea dx,ask1
         call disp                         ;提示输入源文件标识符
         lea dx,sfile
         call inpt                         ;接收源文件标识符
         mov cl,sfile+1
         xor ch,ch
         mov si,cx
         mov [si+sfile+2],0                ;在(文件标识符)字符串后加 00H 字节
         lea dx,ask2
         call disp
         lea dx,dfile
         call inpt                         ;接收目标文件标识符
```

```
        mov cl, dfile+1
        xor ch,ch
        mov si,cx
        mov [si+dfile+2],0
        lea dx,note
        call disp                    ;提示插入磁盘
        mov ah,7
        int 21h
        call copy
        ret
start   endp
disp    proc near
        mov ah,9
        int 21h
        ret
disp    endp
inpt    proc near
        mov ah,0ah
        int 21h
        ret
inpt    endp
copy    proc near
        mov ah,3ch
        lea dx,dfile+2
        mov cx,0020h
        int 21h                      ;建立目标文件
        lea dx,er1
        mov bx,ax
        jc err
        mov bufr,ax
        mov ah,3dh
        mov al,0
        lea dx,sfile+2
        int 21h                      ;打开源文件
        lea dx,er2
        mov bx,ax
        jc err
r_w:    mov cx,0010h
        mov ah,3fh
        lea dx,sfile+2               ;读源文件
        int 21h
        lea dx,er3
        jc err
```

```
        or ax,ax
        je exit
        mov ah,40h
        lea dx,sfile+2
        xchg bufr,bx
        int 21h                    ;写入目标文件
        lea dx,er4
        jc err
        xchg bufr,bx
        jmp r_w
exit:   mov ah,3eh                 ;正常结束,关闭文件
        int 21h
        lea dx,er5
        jc err
        xchg bufr,bx
        mov ah,3eh
        int 21h
        lea dx,er6
        jc err
        ret
err:    mov ah,3eh                 ;出现错误,关闭文件
        int 21h
        xchg bufr,bx
        mov ah,3eh
        int 21h
        call disp                  ;错误提示
        ret
copy    endp
code    ends
        end     start
```

2.5.3 磁盘接口实验

实验 2.6 用 FCB 方法复制文件

实验要求：

1. 将一个指定文件复制成另一个指定文件。
2. 复制文件的过程：

(1)建立目标文件；

(2)打开源文件；

(3)读源文件；

(4)写入目标文件；

(5)关闭目标文件和源文件。

3. 文件复制结束后，提示正确或错误。

实验 2.7 用文件代号方法分页显示文件

实验要求：

1. 分页显示指定文件的内容。

2. 分页显示文件的过程：

(1)输入文件名；

(2)打开该文件；

(3)从文件读出并显示一页；

(4)按任意键显示下一页；

(5)关闭文件。

实验 2.8 BIOS 读磁盘根目录

实验要求：

1. 用 BIOS 的读磁盘功能调用直接从磁盘扇区中读目录并放入内存中。

2. 在屏幕上显示目录，每行 4 个目录名。

2.6 打印机接口

PC 系列机一般采用 CENTRONIC 并行接口标准来与打印机相连，该并行接口也可用来作为同它的 I/O 能力相匹配的任何设备实现并行通信。

2.6.1 打印机接口方法

打印机接口有两种方法：一是直接面向打印机适配器编程，二是 BIOS 和 DOS 功能调用。

一、对打印机适配器编程

对打印机适配器编程可采用两种控制方式：一是采用查询控制方式；二是采用中断控制方式。

1. 查询控制方式

(1)送打印数据到数据端口(并口 1 为 378H，并口 2 为 278H)。

(2)循环读入打印状态字(并口 1 为 379H，并口 2 为 279H)直到打印机“不忙”。

(3)若“忙”信号为高电平时(不忙)，通过控制锁存器(并口 1 为 37AH，并口 2 为 27AH)输出数据选通信号$\overline{\text{STROBE}}$到打印机，打印机接收数据。

(4)当打印机接收到一行字符时，打印机立即开始打印，此时“忙”信号为低电平，计算机不能继续送出打印和选通信号，必须等打印机打印完毕，方可继续传送数据，直到打印完全部信息。

2. 中断控制方式

(1)暂时禁止中断。

(2)将打印中断服务程序入口作为中断 0FH(IR_7)向量放入中断向量表。

(3)开中断。

(4)等待打印机中断。

(5)若发生打印机中断，说明打印机已准备就绪，执行打印机中断服务程序将打印数据送到打印机。

二、打印机的 BIOS 和 DOS 功能调用

打印机的 BIOS 调用 INT 17H 有 3 个功能号 0～2，其中功能号 0 为打印一个字符，功能号 1 为初始化打印机，功能号 2 为读打印机状态。

打印机的 DOS 系统调用 INT 21H 的功能号 5 为标准打印，该调用只驱动打印机号 1(并口 1)。

2.6.2 应用实例

例 2.8 查询方式打印机输出字符串

打印一行字符 PRINTER DISPLAY PROGRAM

```
;     exam28.asm

data      segment
buf       db ' printer display program'     ;定义数据区中的一行字符
crlf      db 13,10
data      ends
stack     segment stack                     ;定义堆栈区有 128 个字节
          db 128 dup(0)
stack     ends
code      segment                           ;定义代码段
          assume cs:code,ds:data,ss:stack
start:    mov ax,data                       ;设置 DS
          mov ds,ax
          mov bx,offset buf                 ;送待打印字符串首地址
rot:      mov al,[bx]                       ;取一字符至 AL
          mov dx,378h                       ;DX 指向第一打印机适配器数据端口地址
          out dx,al                         ;输出一字符至打印机适配器
          inc dx                            ;DX 指向状态端口地址
wat:      in al,dx                          ;读入状态字
          test al,80h                       ;测试打印机忙否
          jz wat                            ;"忙",转回重新测试等待
stb:      inc dx                            ;"不忙",发选通信号。DX 指向控制端口
          mov al,0dh                        ;控制字中选通位置一
          out dx,al                         ;发选通信号,打印机接收字符
          mov al,0ch                        ;控制字中选通位恢复为 0
          out dx,al
          cmp byte ptr[bx],10               ;测试输出字符是否换行符
          jc next                           ;是换行,转入下一段程序
          inc bx                            ;不是换行,BX 指向下一字符地址
          jmp rot                           ;转回去,输出下一个字符
```

```
next:   mov ax,4c00h                    ;返回操作系统
        int 21h
code    ends
        end     start
```

例 2.9　中断方式打印机输出字符串

打印一行字符 PRINTER DISPLAY PROGRAM

```
;       exam29.asm

data    segment
buf     db ' printer display program' ,13,10
addr    dw 0
data    ends
stack   segment stack
        db 256 dup(0)
stack   ends
code    segment
start   proc  far
        assume cs:code,ds:data,ss:stack
        push ds
        xor ax,ax
        push ax
        mov ax,data
        mov ds,ax
        mov ax,0
        mov es,ax                       ;设置 ES 为 0
        mov ax,stack
        mov ss,ax
        cli                             ;暂时禁止中断
        mov ax,offset intsr ;打印中断服务程序首地址存入中断向量表
        mov es:[3ch],ax
        mov ax,cs
        mov es:[3eh],ax
        sti                             ;重新允许中断
        mov addr,offset buf             ;送打印字符串首地址至 ADDR
        mov al,10                       ;输出一换行字符,引发打印机中断请求
        mov dx,378h
        out dx,al                       ;输出字符
        inc dx
wai:    in al,dx                        ;判打印机是否忙?
        test al,80h
        je wai
        inc dx
        mov al,1dh                      ;发选通,且使中断允许位置 1
```

```
        out dx,al
        mov al,1ch
        out dx,al
        in al,21h                ;使 8259A 不屏蔽打印机中断请求
        and al,7fh
        out 21h,al
        mov bx,7fh
b4:     mov cx,0ffffh            ;等待打印机中断,或执行别的程序段
b3:     dec cx
        jnz b3
        dec bx
        jnz b4
        ret
        start    endp
intsr   proc far                 ;打印中断服务程序入口
        push ax                  ;保护寄存器
        push bx
        push dx
        mov bx,addr              ;送字符地址至 BX
        mov al,[bx]              ;取一字符
        mov dx,378h              ;DX 指向数据端口地址
        out dx,al                ;输出字符
        cmp al,10                ;是换行字符吗?
        jne b1                   ;不是,转去发选通
        in al,21h                ;是换行,屏蔽打印机中断请求
        or al,80h
        out 21h,al
b1:     inc dx
        inc dx                   ;DX 指向控制端口地址
        mov al,1dh               ;发选通,且允许中断
        out dx,al
        mov al,1ch
        out dx,al
        inc addr                 ;字符地址加一,指向下一个字符
        mov al,20h               ;发中断结束命令
        out 20h,al
        pop dx                   ;恢复寄存器
        pop bx
        pop ax
        iret                     ;中断返回
intsr   endp
code    ends
        end      start
```

例 2.10 打印缓冲区字符

通过 INT 17H 的子功能 AH＝0 发送一字符到打印机并利用返回的状态(在 AH 中)判断是否“超时”、“错误”或“纸尽”。若存在某种故障,则相应显示出错信息并等待用户的响应,否则,正常返回。

```
;       exam210.asm

                ;打印缓冲区字符的子程序
data            segment
buffer          db  ' printer display program ' ,0dh,0ah
leng            equ $ -buffer
mesg_1          db 0dh,0ah,' printer out of time. strike any key when ready. $'
mesg_2          db 0dh,0ah,' printer offline. strike any key when ready. $'
mesg_3          db 0dh,0ah,' printer out of paper. strike any key when ready. $'
data            ends
stack           segment para stack ' stack'
                dw 256 dup(?)
stack           ends
code            segment
prt_buf         proc  far
                assume cs:code,ss:stack,ds:data
start:          push ds
                xor ax,ax
                push ax
                mov ax,data
                mov ds,ax
                mov ax,stack
                mov ss,ax
                lea si,buffer
                mov cx,leng
                mov ah,1                ;打印机初始化
                mov dx,0
                int 17h
again:          mov ah,0                ;打印一个字符
                mov dx,0                ;选择 LPT1
                lodsb                   ;取打印字符
                int 17h                 ;输出到打印机
                test ah,00101001b       ;有"超时","错误"?
                jz next                 ;无,打印下一个字符
                test ah,01h             ;是,"超时"?
                jz error                ;不是,继续检测
                lea dx,mesg_1           ;取"超时信息"
                jmp disp                ;转显示
error:          test ah,08h             ;是"错误"?
```

```
                jz paper_out                ;否,转"纸尽信息"
                lea dx,mesg_2               ;取"脱机信息"
                jmp disp                    ;转显示
paper_out:      lea dx,mesg_3               ;取"纸尽信息"
disp:
                mov ah,9                    ;显示错误信息
                int 21h
                mov ah,0                    ;等待用户响应
                int 16h
next:
                loop again
                ret                         ;返回调用者
prt_buf         endp
code            ends
                end     start
```

2.6.3 打印机接口实验

实验 2.9 打印文本文件

实验要求：

1. 分别用查询方式、中断方式和 BIOS 功能调用三种方式编写程序打印指定文件的内容。

2. 分页打印文件的过程

(1)输入打印文件名；

(2)打开文件；

(3)从文件中读出并打印一页；

(4)按任意键打印下一页；

(5)关闭文件。

第三章　用户接口扩展软硬件设计

3.1　用户接口扩展设计方法

3.1.1　微机系统总线

一、概　　述

计算机各部件之间进行信息传输的公用通道称为总线。微型计算机系统中广泛采用总线结构。其优点是系统成本低、组态灵活、维修方便。采用总线标准设计、生产的硬件模块兼容性强，可以方便地组合在一起，通过系统总线交换信息，构成满足不同需要的微机系统。

总线的种类很多。在个人微机系统中，使用最广的是基于 Intel 公司的 X86 微处理器系列的 PC 机中，其底板上使用了 ISA、EISA、VL—BUS、PCI 等总线，与磁盘等高速外部设备联接的有 SCSI 接口。在工业控制微机系统中，使用最广泛的是 STD 总线和 ISA 总线。

二、ISA 总线

1. 概述

IBM-PC/XT 系统的 I/O 扩展槽具有 62 引脚，常称之为标准 PC 总线或 XT 槽。PC/AT 系统的 I/O 扩展槽在 62 引脚槽的基础上，又增设－36 引脚的小槽，通常称之为扩展 PC 总线或 AT 槽。由于 PC 总线已成为 8/16 位数据传输总线的工业标准，故又被命名为 ISA(Industry Standard Architecture)总线。

自 IBM 推出世界上第一台 286 后，AT 机的 ISA 标准就得到了确认。于此同时，以兼容这一标准为前提的微型机产品纷纷出笼，占据了整个微型机市场。目前，人们所见到的 286、386、486 微型机绝大多数是这个标准的，尽管它们的工作频率各异，内部的功能也略有不同，都旨在提高系统的性能，由于受到 ISA 标准的限制，这种性能的提高是非常有限的。

ISA 标准定义了一条系统总线标准，数据宽度为 16 位，工作频率为 8MHz，传输率最高为 8MB/s。众所周知，自从 Intel 公司推出 80386CPU 之后，使系统内部总线结构产生一个飞跃的变化，数据总线宽度由 16 位增到 32 位，CPU 处理能力大大提高。但由于 ISA 标准限制，系统总的性能没有根本改变。凡是系统总线上的 I/O，存储器的访问也不会有很大改进，因而使得在强大的 CPU 处理能力与低性能的系统总线间形成了一个瓶颈，制约了机器性能的提高。为了打破这一瓶颈，IBM 公司在推出它的第一台 386 微机时，突破了传统的 ISA 标准，创造了一个全新的与 ISA 标准完全不同的系统总线标准——MCA(Micro Channel Architecture)标准，即所谓的微通道结构。该标准定义了系统总线上的数据宽度为 32 位，并提供了 burst Mode 使得数据的传输率是 ISA 的 4 倍，为了保持 IBM 公司在微型机领域的领导地位，该公司没有将这一标准公诸于众，以求垄断市场，这使得解决瓶颈问题的手段为 IBM 公司所独有。随着 Intel 公司 80486CPU 的推出，对于解决“瓶颈”问题的需求日益增加，为了冲破 IBM 公司 MCA 封锁，1989 年以 Compaq 为代表的包括 HP、AST、EPSON、NEC、Olivetti、

Tandy、Wyse 以及 Zenjth Datn System(ZDS)九家公司联合推出了一个新的系统总线标准——EISA(Extended Industrial Standard Architecture)。EISA 不仅具有 MCA 的全部功能,同时还保持了与传统的 ISA 百分之百兼容,这一点 MCA 是做不到的。由于 EISA 的开放性,到目前为止,已有上百种 EISA Add-on 卡包括 LAN、SCSI、IDE、Graphic 等相继问世,使 EISA 在应用领域得到充分发展。

EISA 是从 ISA 发展而来的,同 ISA 相比 EISA 系统主要具有以下几个特点:

(1)较高的 I/O 性能:32 位数据总线宽度;33MB/s 数据传输率;多总线主控(Bus Master)。(2)Add-on 卡的安装十分容易:自动配置,无需 DIP 开关。(3)维护 Add-on 卡的投资:保持百分之百的 ISA 兼容。

EISA 还可支持多总线 Master 和对总线 Master 的智能管理。最多支持 6 个总线 Master。

2. PC 系列微机 I/O 扩展槽的外型结构

XT 扩展槽的外型结构如图 3.1,它的 62 个引脚分布在两面,分别标注为 A_1～A_{31}和 B_1～B_{31},其中 A 面对应扩展卡的元件面,B 面对应扩展卡的焊接面,引脚的标号从机箱后部起始。引脚间距(中心相距)为 1/10 英寸,即 2.54mm。扩展槽可插扩展卡的厚度一般为 1.5mm。一台标准的 PC/XT 共装有 8 个 I/O 扩展槽,分别标注为 J_1～J_8,槽编号从左侧起始依次向右。

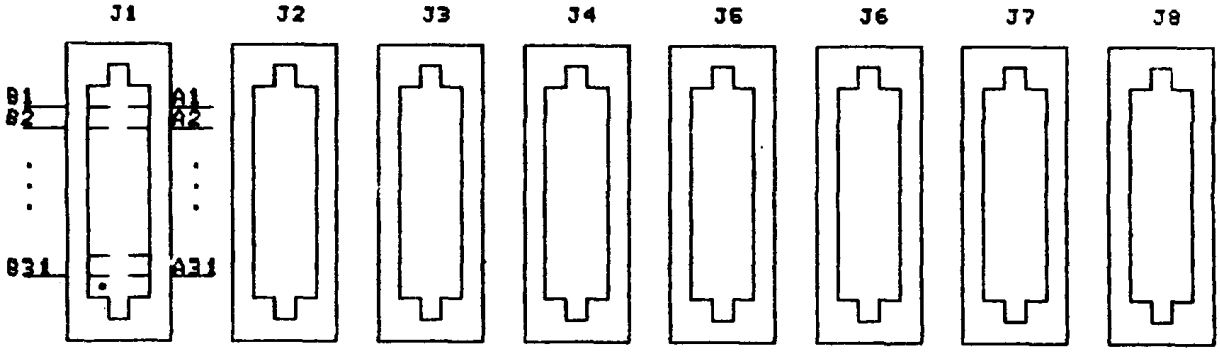


图 3.1 XT 槽的外型结构

AT 扩展槽的外型结构如图 3.2,它的 62 引脚的大槽与 PC/XT 完全一致,32 引脚的小槽与 62 引脚的大槽并列对齐,每边引脚排列分别为 C_1～C_{18}和 D_1～D_{18},其中 C 面与 A 面同在一边,D 面与 B 面同在一边,引脚的标号从靠紧大槽端起始。一台标准的 PC/AT 共装有 2 个 XT 槽和 6 个 AT 槽。

3. ISA 总线信号线定义

在 IBM-PC/XT 的 I/O 扩展槽 J_1～J_8 中,同号引脚都具有相同的名称和特性(只有 J_8 槽 B_8 引脚为一输入信号,J_1～J_7 槽该处为空),因为系统总线的 62 个信号分别引到 8 个 62 引脚的插槽上。这 62 个信号包括时钟与复位信号,20 位地址线,8 位双向数据线,存储器和 I/O 读写控制线,6 个中断请求线,3 个 DMA 通道控制级,4 组电源(±12V,±5V)和地线等。所有信号电平均为 TTL 电平。

XT 槽内的信号定义如图 3.3。各信号的名称和作用如表 3-1 所示。

在 AT 槽中,62 引脚的大槽其信号排列与功能基本与 XT 槽相同,只有两点改动:一是 B_{19}由原为 DMA 通道 0 的响应线改为指示 RAM 刷新周期$\overline{\text{REF RESH}}$;二是 B_8 由原仅有 J_8 上是板选中信号,现改为各个槽都接有零等待状态 0WS 信号。另外,还有系统时钟周期不同,它为 167ns,占空比 50%。36 引脚的小槽内信号的排列如图 3.4,各信号的名称和作用如表

3-2 所示。

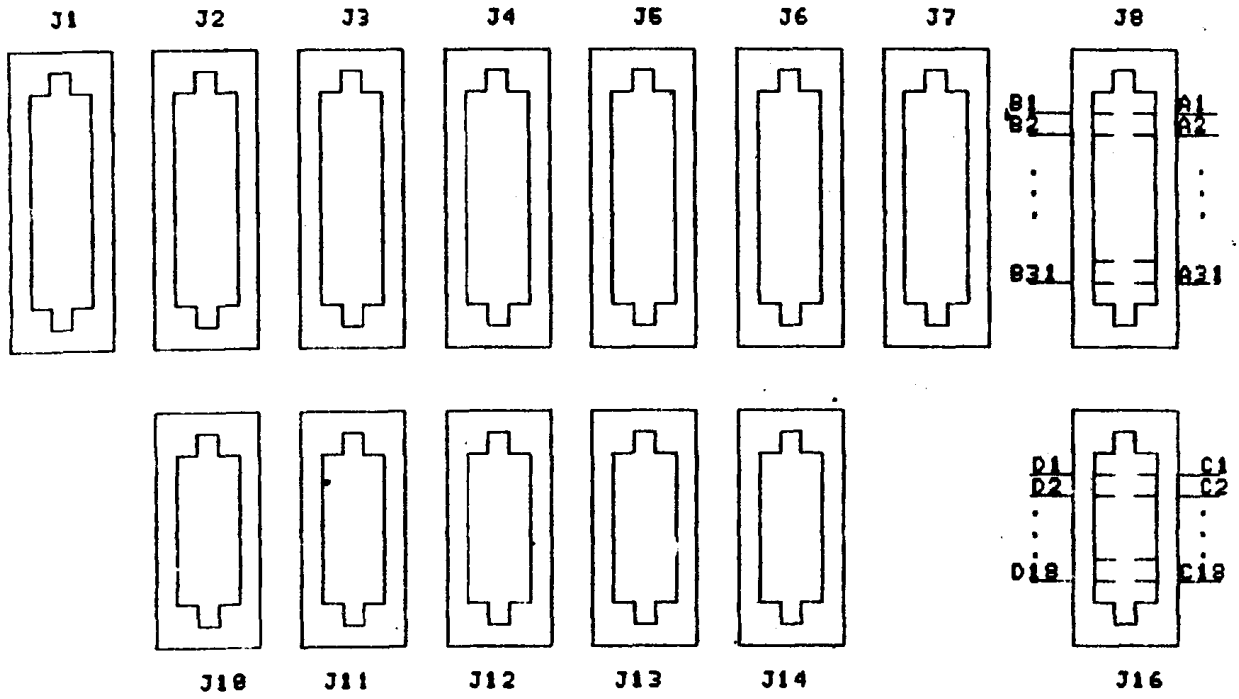


图 3.2 AT 槽的外形结构

表 3-1

符 号	名 称	I/O	作 用
OSC	主振时钟	O	频率为 14.318MHz(70ns),其占空比为 50%
CLK	系统时钟	O	它是由主振频率三分频而来,时钟频率为 4.77MHz(周期 210ns),占空比为 33%
Reset	系统复位	O	用于系统复位或初始化系统逻辑
ALE	允许地址锁存	O	此信号由 8288 总线控制器产生,用于系统上锁存由 CPU 送出的有效地址,它也可用于 I/O 扩展槽作为 CPU 有效地址指示(当与 AEN 一起作用时),CPU 地址在 ALE 的下降沿被锁存。
A_0～A_{19}	地址总线	O	用于对存储器和 I/O 部件进行寻址,其中 A_0 为最低位,A_{19} 为最高位
D_0～D_7	数据总线	I/O	用于对 CPU、存储器和 I/O 部件提供 8 位数据,其中 D_0 为最低位,D_7 为最高位。
$\overline{MEMR}$	存储器读	O	它指示存储器的数据送上数据总线
$\overline{MEMW}$	存储器写	O	它指示数据总线上的数据送入存储器存储
$\overline{IOR}$	I/O 读	O	它指示一个 I/O 部件的数据送上数据总线
$\overline{IOW}$	I/O 写	O	它指示一个 I/O 部件读取数据总线上的数据
IRQ_2～IRQ_7	中断请求 2～7	I	用于通知 CPU 为某个 I/O 部件服务,其中 IRQ_2 优先级最高,IRQ_7 优先级最低。
AEN	地址允许	O	用于阻止 CPU 控制总线,而允许 DMA 操作,当该信号为高电平时,由 DMA 控制器控制系统总线。

DRQ_1～DRQ_3	DMA 请求 1～3	I	外设发出该信号，用于请求 DMA 操作，它的优先级排队以 DRQ_1 最高，以 DRQ_3 最低。
$\overline{DACK_0}$～$\overline{DACK_3}$	DMA 响应 0～3	O	$\overline{DACK_1}$～$\overline{DACK_3}$用于响应 DRQ_1～DRQ_3 的回答，$\overline{DACK_0}$用于刷新系统动态 RAM
T/C	终结计数	O	在任何一个 DMA 通道计数终结时，由 DMA 控制器发出一个正脉冲
$\overline{CARD\ SLCTD}$	扩展板选中	I	仅用于扩展槽 J_8 中的扩展板。由扩展板输入，它向母板指明：该扩展板已选中。
$\overline{I/O\ CHCK}$	I/O 通道检查	I	该信号为 CPU 提供扩充存储器或 I/O 部件上奇偶错误信号，当信号为低电平时，指出一个奇偶校验出错
I/O CHRDY	I/O 通道就绪	I	该线通常为高(准备好)。仅当 I/O 部件或存储器需延长 I/O 或存储器周期时可使其变低(未准备好)，此时，I/O 或存储器周期以时钟周期的整倍数扩展。注意，当一个慢速 I/O 部件或存储器在检测到一个有效地址和一个读或写命令时，立即使该线信号变低，但该线信号的低电平维持时间不得超过 10 个时钟周期。
+5V	+5V 直流电源	O	接到两个脚上，供扩展板使用
−5V	−5V 直流电源	O	供扩展板使用
+12V	+12V 直流电源	O	供扩展板使用
−12V	−12V 直流电源	O	供扩展板使用
GND	地线	O	接到三个脚上，供扩展板使用

表 3.2

符　号	名　称	I/O	作　用
LA_{17}～LA_{23}	可锁存地址总线	O	最高 7 位地址线，其中 LA_{23}为最高位
D_8～D_{15}	数据总线	I/O	为 CPU 存储器和 I/O 部件提供高 8 位数据，其中 D_{15}为最高位
BHE	高位总线允许	I/O	高 8 位数据允许信号
$\overline{MEMCS16}$	存储器片选	I	16 位存储器片选信号
$\overline{I/O\ CS16}$	I/O 片选	I	16 位 I/O 片选信号
$\overline{MEMR}$	存储器读	I/O	用于对所有存储器的读(大槽中的此信号只能用于 1MB 空间)
$\overline{MEMW}$	存储器写	I/O	用于对所有存储器的写(大槽中的此信号只能用于 1MB 空间)
IRQ_{10}～IRQ_{12} IRQ_{14}～IRQ_{15}	中断请求 10～12、14～15	I	用于请求 CPU 为某个 I/O 部件服务，其中 IRQ_{10}优先级最高，IRQ_{15}优先级最低
DRQ_0 DRQ_5～DRQ_7	DMA 请求 0、5～7	I	外设用于请求 DMA 操作，其优先级 DRQ_0 为最高，DRQ_7 为最低
$\overline{MESTER}$	总线主控	I	外设用于控制系统总线处于三态，以便主控总线
+5V	+5V 直流电源	O	供扩展板使用
GND	地线	O	供扩展板使用

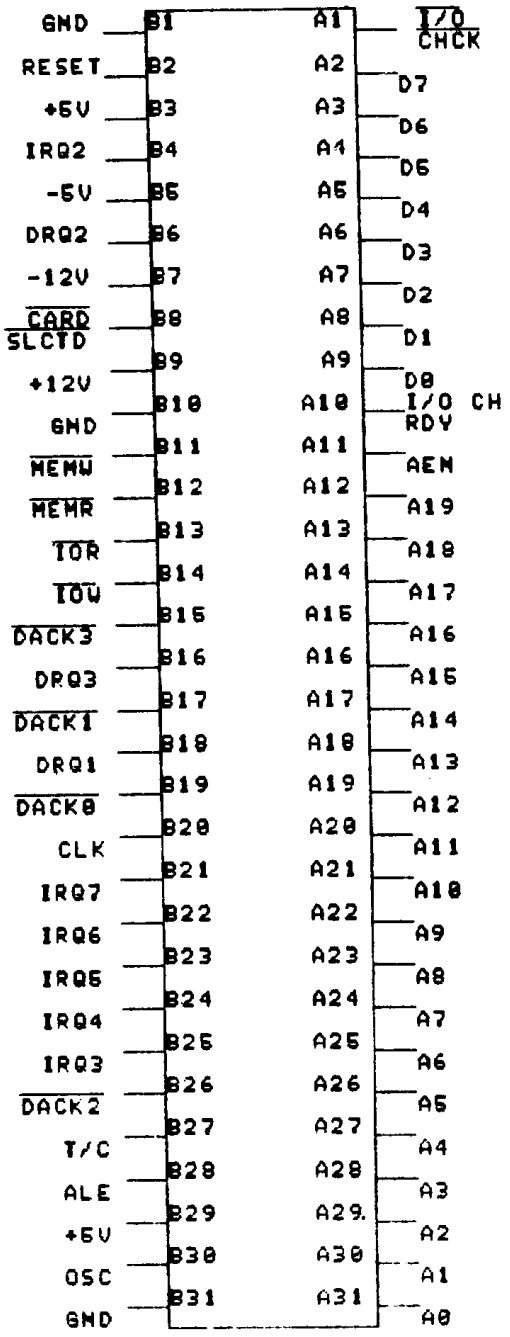


图 3.3 XT 槽内的信号定义

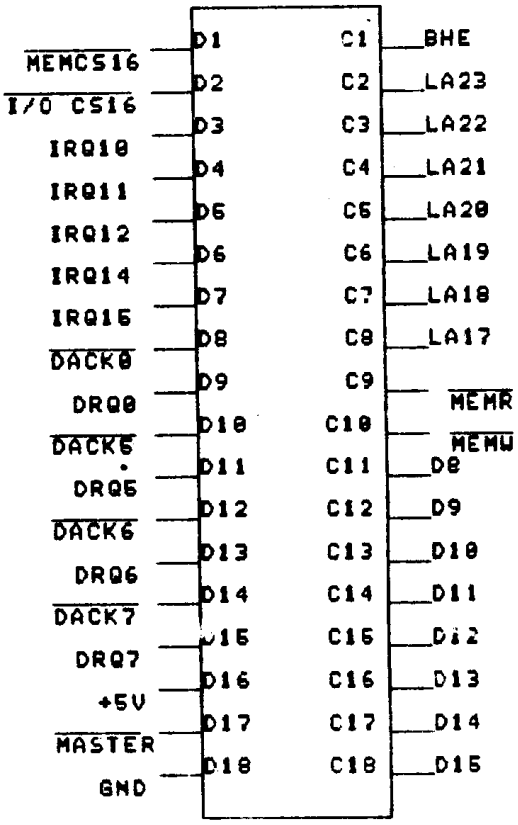


图 3.4 AT 槽内的信号定义

4. ISA 总线的负载与驱动能力

设计一个应用系统时，总线信号的驱动能力与总线负载是一项重要的考虑因素。对于输出到总线的信号，你必须计算你的设计是否有足够的驱动能力。对于由总线输入到应用系统的信号，你必须考虑它是否有足够推动能力来驱动你的应用系统。

(1)总线的驱动能力

总线输出信号有两个规格较重要，即 IOL 与 IOH。IOL 表示在低电位时输出驱动器所能流入(Sink)的最大电流；IOH 则是高电位时由驱动器流出的最大电流。表 3-3 列出 I/O 扩展槽驱动器的驱动能力。对一片用户卡的实际驱动能力则是由表中之值扣掉其余 I/O 卡上所

消耗者。

表 3-3 系统总线驱动能力

汇流排输出信号	IOL	IOH
$D_0 \sim D_7$	23.6	−14.96
$A_{19} \sim A_{16}$	7.2	−2.46
$A_{14} \sim A_{15}$	21.2	−2.51
A_{13}	23.2	−2.56
$A_0 \sim A_{12}$	23.4	−2.56
IOR, IOW, MEMR, MEMW	23.8	−4.98
CLK	23.2	−14.96
AEN	24.0	−15.00
$DACK_0$	24.0	−15.00
$DACK_1$	3.2	−0.20
$DACK_2$, $DACK_3$	2.8	−0.18
ALE	14.8	−0.94
RESET DRV	8.0	−0.40
T/C	8.0	−0.40
OSC	5.0	−1.00

注：(1)所有电流单位均为 mA。

(2)表中之值随主机板的负载而改变。

(3)均为 8088 汇流排驱动或 DMA 汇流排驱动之“最差情况”的值。

2. 电容性负载因素

I/O 扩展槽总线输出信号的电容性负载是应用系统设计上的另一项考虑的因素。因为每增加一个负载，驱动线路所见的电容也将增加。电容越大，信号就会失真与延迟。每个负载约增加 10～20pF 的电容到总线上，十个负载就达 100～200pF 之值。电容值若大于 200pF，其输出信号常受到影响，而可能造成系统不稳定。

三、STD 总线

1. 概述

1978 年，STD 总线最初设计完成并介绍给世人之时，其目标是和所有 8 位微处理器相兼容，是以小尺寸(4.5in×6.5in)、高可靠性、低价格的面貌出现，为嵌入式系统提供解决手段。几乎遍及整个工业领域的丰富 I/O 接口，加上 IEEE 将它作为 961 标准正式推出，使得 STD 总线成为为人熟知的工业标准，在世界范围已安装了大约 50 万套系统。通常人们把 STD 总线称为“兰领总线”，因为它把侧重点放在工业测控应用上。

16 位微处理器问世后，它采用总线复用和周期窃取的办法，可以和所有 16 位微处理器完全兼容。

32 位微处理器发展起来以后，STD 总线一方面可以采用扩展的 STD32 总线的方法，另一方面采用 All-in-one 的方式可以和 32 位微处理器兼容。All-in-one 设计方法的推动力是：

由于 VLSI 技术发展导致了高性能、高集成度芯片的不断涌现；表面安装技术(SMT)及片状元件的发展；门阵列技术的发展，如 GAL、PAL 以及高性能 PLD；ASIC 技术的发展。

这些技术的发展导致了在一块小板上也能集成非常强大的功能，由此，小板愈来愈显示出其优越性。例如，采用 All-in-one 模式可以将一台 IBM PC386 或 486 集成在一块 STD 总线小板上，这种 All-in-one 的设计方法，使得原来的系统总线转变为 I/O 总线，消除了总线的瓶颈，这是 8 位只有 56 条信号线的 STD 总线之所以能与 32 位微处理器兼容的主要原因。

All-in-one 模式是当前工控机平台系统的流行趋势，IPC 和 PC/104 均是如此，为此：新型 STD 总线设计基于 16 位和 32 位微处理器单元，比如 8088，80188，V40，80186，80286，80386SX/DX，80486SX/DX 和 68030。随着软件开发工具、操作系统和应用程序性能的提高，一个新的应用领域正在逐渐形成。尽管先前基于 Z80，6801 和 8085 设计的 8 位 CPU 设计已经过了高峰期，但广大用户还是需要提供稳定产品。因此，这些老产品仍然在批量生产。特别是国产 STD 总线工控机中，以 MSC-51 单片机组成的系统销量最多。

STD 总线所取得的新的进展包括与 PC/XT 和 PC/AT 兼容，16 位数据传输，ARC net 和以太网联接，价格非常低的单板计算机，全 CMOS 工艺允许较宽温度范围工作，智能 I/O 卡，以及高速图形技术等等，就构成了所谓"新型"STD 总线。这个"新型"总线充满了生机，具有蓬勃发展的势头，一个与 DOS 完全兼容的嵌入式操作系统，开发工具，实用程序和应用软件的全新领域已成为现实，即所谓"新型"STD 总线正在演变成真正的工业 PC。

STD 总线已经使用到 80186DX，以及高集成度的支持芯片系列和 BIOS，以期为工业应用提供完整丰富的 I/O 卡。重要的因素在于为现有的模板提供一条改进升级的渠道。STD 总线这样做的结果，使得它能持久地作为工业应用标准，而不是每隔 5 年就受到被摒弃的威胁。开放式结构、高强度工业设计、小尺寸、低价格、简便的 I/O 接口，总之，STD 的大众化从已安装的 50 多万套 STD 总线系统中得到了极好的证明。

2. STD 总线的机械特性

模板的几何尺寸应符合图 3.5 所示的规定，56 引脚(双 28 脚)插头的尺寸如图 3.6 所示。引脚中心距为 0.125 英寸约 3.18 毫米。

模板上元器件的高度没有严格规定。但要求相邻板之间的最小间隔(前一块板上引脚尖端到后一块模板元器件凸出部分之间)至少为 0.01 英寸即 0.25 毫米。

目前国内各厂出售的机笼、母板槽间距有 5/8 英寸、6/8 英寸和 1 英寸三种。可以根据模板元器件高度选用。必要时可以一块模板占二个槽的空间(隔开一槽再插另一块模板)。

母板有普通的双面板和高性能的四层母板。四层母板是在元件面、焊接面之间增加了接地层和电源层。使每两根信号线之间都有地线隔离，多点接地，既降低了线路阻抗又提高了抗干扰能力。母板上的插槽数有 6，9，13，17，21 等多种，用户可根据系统所用模板数选用。

3. STD 总线信号线定义

STD 总线开始推出时，是针对当时的 8 位微型计算机的，随着技术的发展和应用的需要，STD 总线经过修订和改进，利用复用技术，在原定义的 56 个总线信号之下，实现了 STD 总线支持 20 位地址，寻址 1 兆字节的直接寻址能力。在保证同现有 I/O 插件板兼容的条件下，提供全 16 位数据的传送能力。目前，国内所采用的大多是这种 8 位，16 位兼容的 STD 总线。80 年代末，STD 总线也由 56 个信号发展到 114 个信号。目前，136 个信号的 32 位 STD 总线标准已经推出，为高档的 STD 微机系列的发展提供了有利的条件。下面，我们就以国内

最流行的 56 个信号的 STD 总线为例加以说明。

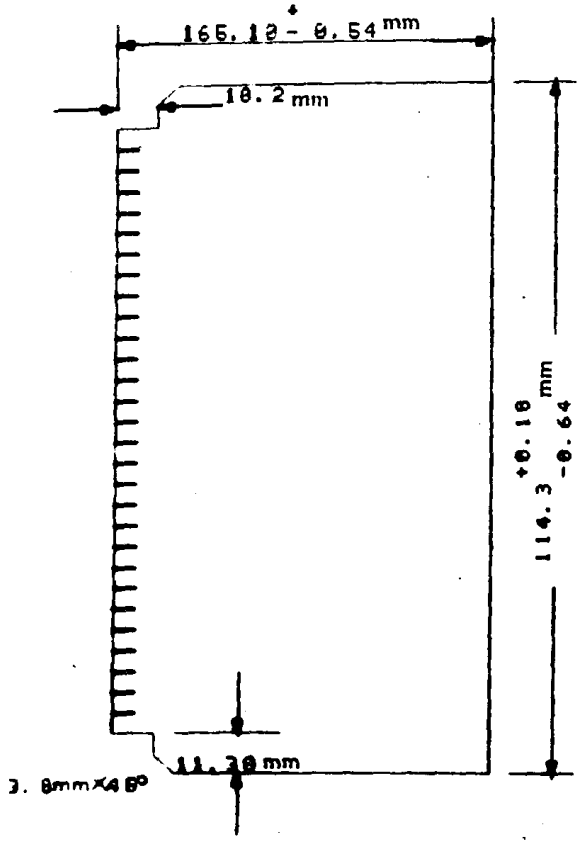


图 3.5 STD 总线模板外形结构

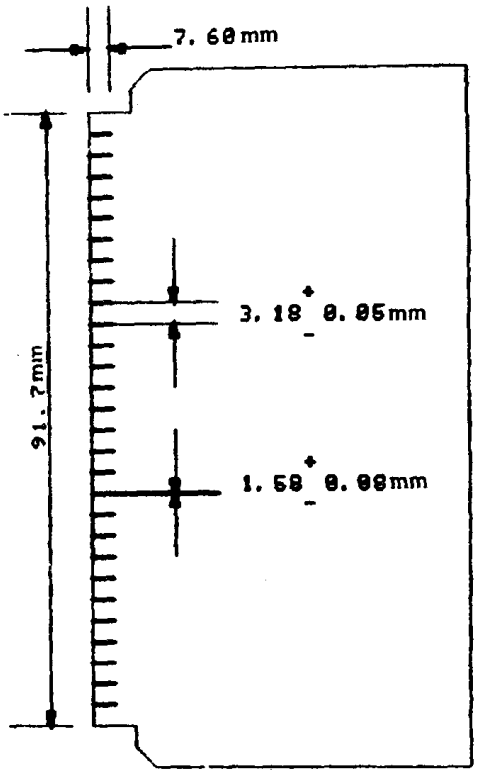


图 3.6 STD 模板插头结构

(1)引脚分配

STD 总线有 56 个引脚，分为 5 个功能组，各组的引脚表示为：

引脚 1～6：逻辑电源总线

引脚 7～14：数据总线

引脚 15～30：地址总线

引脚 31～52：控制总线

引脚 53～56：辅助电源总线

STD56 引脚的详细分配如表 3-4 所示，表中信号流向以现行主设备为参考点。

(2)信号说明

① 电源。STD 总线定义引脚 1～6 和 53～56 为电源和地。如表 3-4 所示，包括±5V 和±12V。其中引脚 1，2 为＋5V 逻辑电源；引脚 3，4 为逻辑地；引脚 5，6 是双重定义，可以接－5V，又可以分别用作备用电池供电和掉电信号的引脚。53，54 为辅助地而 55 和 56 引脚分别定义为＋12V 和－12V 辅助电源。

② 数据总线。这是双向、三态工作的总线。在我们定义的信号中，若是 8 位处理器，存储器的地址范围在 64K 字节之间时，不需要对数据总线进行复用。若地址范围大于 64K 字节时，需利用机器周期同步信号 MCSYNC*，复用数据总线，利用它传送高位地址(A_{16}～A_{23})。

③ 地址总线。STD 定义最多达 24 条地址总线(A_0～A_{23})。当地址寻址范围超过 64K 字节时，需复用数据总线作为地址总线使用。同样，当数据超过 8 位时，例如用全 16 位数据传

送时，又要复用地址总线，用它来传送数据的高 8 位。

表 3-4 STD 总线信号分配表

	元件面				焊接面			
	引脚	信号名	信号流向	说明	引脚	信号名	信号流向	说明
逻辑电源总线	1	V_{CC}	输入	逻辑电源+5V	2	V_{CC}	输入	逻辑电源+5V
	3	GND	输入	逻辑接地	4	GND	输入	逻辑接地
	5	VBB·1 /VBAT	输入	逻辑偏置·1 /电池	6	VBB·2 /DCPD	输入	逻辑偏置·2 /掉电电源
数据总线	7	D_3/A_{19}	输入/输出		8	D_7/A_{23}	输入/输出	
	9	D_2/A_{18}	输入/输出	数据总线/地	10	D_6/A_{22}	输入/输出	数据总线/地
	11	D_1/A_{17}	输入/输出	地扩展总线	12	D_5/A_{21}	输入/输出	地扩展总线
	13	D_0/A_{16}	输入/输出		14	D_4/A_{20}	输入/输出	
地址总线	15	A_7	输出	地址总线	16	A_{15}/D_{15}		地址总线/数据扩展总线
	17	A_6	输出		18	A_{14}/D_{14}		
	19	A_5	输出		20	A_{13}/D_{13}		
	21	A_4	输出		22	A_{12}/D_{12}		
	23	A_3	输出		24	A_{11}/D_{11}		
	25	A_2	输出		26	A_{10}/D_{10}		
	27	A_1	输出		28	A_9/D_9		
	29	A_0	输出		30	A_8/D_8		
控制总线	31	WR*	输出	写	32	RD*	输出	读
	33	IORQ*	输出	IO 地址选择	34	MEMRQ*	输出	存储器地址选择
	35	IOEXP*	输入/输出	IO 扩展	36	MEMEX*	输入/输出	存储器扩展
	37	REFRESH*	输出	刷新定时	38	MCSYNC*	输出	CPU 周期同步
	39	STATUS1*	输出	CPU 状态	40	STATUS0*	输出	CPU 状态
	41	BUSAK*	输出	总线响应	42	BUSRQ*	输入	总线请求
	43	INTAK*	输出	中断响应	44	INTRQ*	输入	中断请求
	45	WAITRQ*	输入	等待请求	46	NMIRQ*	输入	非屏蔽中断请求
	47	SYSRESET*	输出	系统复位	48	PBRESET*	输入	按钮复位
	49	CLOCK*	输出	处理器时钟	50	CNTRL*	输入	辅助定时
	51	PCO	输出	优先级链输出	52	PCI	输入	优先级链输入
辅助电源总线	53	AUXGND	输入	辅助接地	54	AUXGND	输入	辅助接地
	55	AUX+V	输入	辅助电源+12V	56	AUX−V	输入	辅助电源−12V

注：* 表示低电平有效。

① 控制总线。STD 定义了许多控制信号，这也是 STD 总线具有较高性能所需要的。其中有些信号是我们所熟悉且很容易理解，下面就不准备做仔细介绍，只说明那些我们认为重要的信号。

对于用于存储器和 I/O 设备的读写控制的 RD*、WR*、IORQ*、MEMQ*、IOEXP*、MEMEX* 这 6 个信号不再作说明，最后两个信号在后面再进行叙述。

REFRESH* 为动态存储器刷新控制信号。有的 CPU 可以提供这个信号，无此信号的系

统，要由系统设计者自己设计产生。当然，只用静态存储器的系统不需要这个信号。

MCSYNC* 为机器周期同步信号。该信号由现行主设备产生，每个机器周期中出现一次。此信号的性质及定时与处理器有关，用于保持指定的外设与处理器的操作同步。在存储器操作期间，其上升沿用来锁存高位地址 A_{16}～A_{23}，也用来锁存数据总线上复用的地址信号。

STATUS1* 和 STATUS0* 为两个状态控制信号。由现行主设备产生，用以向外围设备提供辅助定时。

上述 4 个总线信号，有的可由采用的 CPU 直接产生。如果 CPU 不能直接提供这些信号，则需要 STD 的设计者自己设计产生。表 3-5 提供了一些 CPU 的外设控制信号。

表 3-5　一些微处理器的外设定时控制线

处理器型号	REFRESH*	MCSYNC*	SYATUS1*	SYATUS0*
8085	—	ALE*	M_1^*	—
Z80	REFRESH*	(RD+WR+INTAK)*	M_1^*	—
8088	—	ALE*	DT/R*	SSO*
6809E	—	Φ_2^*	LIC*	R/W*
6502	—	Φ_2^*	SYNC*	R/W*
68000	—	AS*	M_1^*	SUPE*/USER
8086	—	ALE*	DT/R*	SSO*

中断和总线控制线允许实现 DMA、多重处理、单步、低速存储及电源故障启动等，它们有如下这些信号：

BUSRQ* 和 BUSAK* 总线请求及总线响应信号。

INTRQ* 和 INTAK* 中断请求及中断响应信号。

NMIRQ* 非屏蔽中断请求信号。

WAITRQ* 等待请求信号。可由任何主设备或从设备产生，只要此信号有效，就会使主设备插入等待状态，例如，可用它来实现慢速外设、慢速存储器及单步操作等等。

STD 总线中，控制总线用于时钟和复位的信号为：

SYSRESET* 由加电或系统复位按钮产生的复位信号。

PBRESET* 由输入系统的按钮产生的复位信号，其作用与 SYSRESET* 相同。

CLOCK* 处理器时钟信号，由永久主设备经缓冲提供到总线上，用作系统同步或一般的时钟源。

CNTRL* 辅助定时信号，由专门的时钟定时辅助电路产生，用作实时钟信号或外部输入信号使用。

PCO 和 PCI 是优先级链控制信号，它们均为高电平有效，用以建立中断优先链。

三、ISA 总线与 STD 总线的转换

当前的个人微机系统几乎全采用 ISA 总线，在工业控制系统中常采用 STD 总线。为了两种系统之间的联接，常需要 ISA 总线与 STD 总线之间的转换。本章所提供的所有接口实验模块均采用 STD 总线，目的在于使读者不仅熟悉 ISA 总线系统，而且熟悉 STD 总线系统。实现 ISA 总线到 STD 总线之间的转换有很多种方法。

最简单的一种方法是通过适当的逻辑转换和总线驱动直接将 PC 总线与 STD 总线连通。但由于 PC 机留给用户的可用资源相当有限，采用这一方案将带来许多其他的问题，因此，本书采用 74LS373 锁存的办法，其原理图如图 3.7 所示。

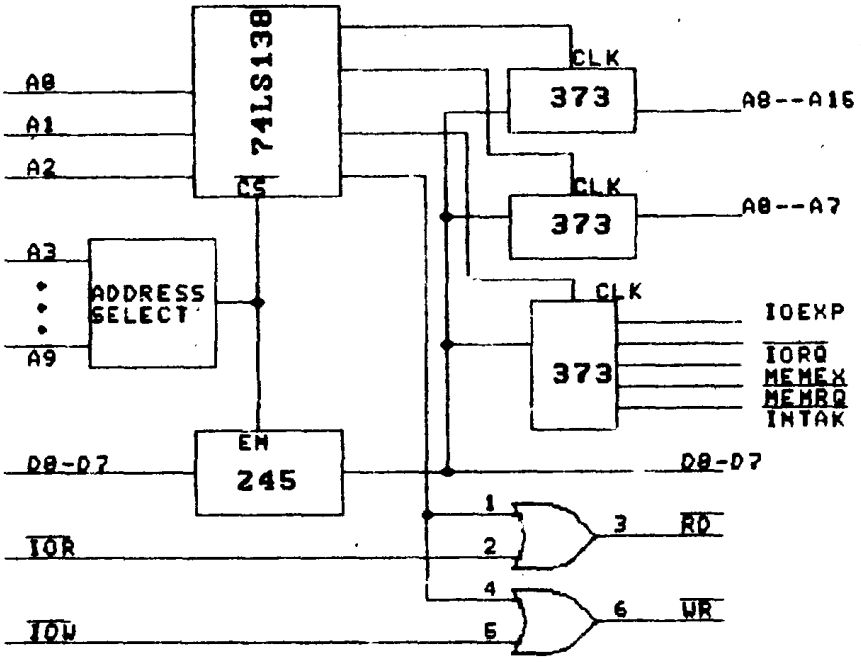


图 3.7　PC/STD 总线转换原理框图

从这张图可以看到，用户通过拨动 8 位 DIP 开关，指定本模块的 I/O 地址(A_3～A_9)，地址比较器产生模块选通信号后，送至 138 和 245 的使能端。A_0，A_1，A_2 参与 138 译码，译码输出与$\overline{IOR}$信号一齐共同产生各个 373 的锁存信号。D_0～D_7 送至 245 的数据输入端，与锁存信号配合，3 个 373 分别锁存 A_8～A_{15}，A_0～A_7 及某些控制信号。245 同时作为数据总线驱动。假定模块高位地址为 1100000(A_9～A_3)，则本模块共占用 4 个 I/O 地址，分别为 300H～303H，当用户要送出 A_8～A_{15}时，使用汇编语言将需要下面这几条指令：

```
MOV DX,300H
MOV AL,高八位地址线数据
OUT DX,AL
```

此时，第一个 373 将高八位地址数据锁存。

由于 303H 地址选通信号控制了读写信号的输出，地址及控制信号准备好后，在读写 303H 口时才送出读写信号，此时才能送出或读入有效的数据：

```
MOV DX,303H
MOV AL,数据线数据
OUT DX,AL
```

下面我们举一个比较完整的例子：

仍然假定 PC/STD 模块高位地址为 1100000，我们在 STD 模块上需要执行一条将 55H 送至 FF21H 口的指令，使用 PC/STD 模板，在 PC 机上利用汇编语言将需要下面一段程序：

```
MOV DX,300H
```

```
MOV AL,0FFH
OUT DX,AL
MOV DX,301H
MOV AL,21H
OUT DX,AL
MOV DX,302H
MOV AL,0EFH
OUT DX,AL
MOV DX,303H
MOV AL,55H
OUT DX,AL
MOV DX,302H
MOV AL,0FFH
OUT DX,AL
```

302H 地址选通信号选中一个 373，完成锁存几个控制信号(IOEXP，$\overline{\text{IORQ}}$，MEMEX，$\overline{\text{MEMRQ}}$，$\overline{\text{INTAK}}$)的功能，对于 STD 模板的 I/O 操作，我们只须送出 IORQ 信号即可(低电平有效)。当 I/O 操作完成之后，应立即将 IORQ 信号关闭，以避免其他的误操作。

3.1.2 总线接口电路设计

总线接口电路实现信号间的组合以满足用户扩展卡的信号线功能及定时要求，接口电路也是实现总线驱动能力的需要。总线接口电路主要由地址译码电路和总线驱动电路组成。

一、I/O 地址译码电路设计

1. I/O 地址分配

在 PC 系列微机中，存储器和 I/O 的地址空间是独立的，存储器的地址空间很大，最多可达 1MB 或 16MB，而 I/O 的地址空间只有 1KB(地址编号从 000H～3FFH)。由于 I/O 接口一部分分布在主机母板上，另一部分分布在扩展槽上，所以 I/O 的 1024 个口地址也分成两部分，其中前 512 个由母板上的接口控制器使用，后 512 个供扩展槽内的接口控制卡使用。

供 I/O 扩展槽内插入接口控制卡使用的 512 个 I/O 地址分配如表 3-6。用户设计开发的 I/O 接口扩展卡只能使用空余的地址。

2. I/O 地址译码电路设计

在用户扩展卡设计中对 I/O 地址译码最简单的方法就是查看 I/O 地址分配表，找出一段未使用到的地址，然后建立适当的译码电路。

地址译码电路可以用门电路和比较器来设计，也可以用专门用于译码的中规模集成电路，如 74LS138，74LS154 等。为了提高使用的灵活性和提高软件的保密性，目前采用 ROM 作为地址译码器的情况也日益增多。

图 3.8 是固定 I/O 地址译码电路设计的例子。

在此例中 74LS138 译码出 2F0H 开始的四个地址。译码信号线再与$\overline{\text{IOR}}$及$\overline{\text{IOW}}$信号相与，产生输入/输出的选通信号。总线信号 AEN 用以抑阻译码，避免在 DMA 周期中作不应该的地址译码。

表 3-6

地址(H)	P/XT 中的分配	PC/AT 中的分配
200～20F	游戏控制卡	游戏控制卡
210～21F	扩展部件	保留
220～2F7	保留	保留
2F8～2FF	串行口控制 2	串行口控制 2
300～31F	试验卡	试验卡
320～32F	硬驱控制卡	
330～36F	保留	保留
370～37F	并行口控制卡 1	并行口控制卡 1
380～38F	同步通信卡 2	同步通信卡 2
390～39F	保留	保留
3A0～3AF	同步通信卡 1	同步通信卡 1
3B0～3BF	单显/打印控制卡	单显/打印控制卡
3C0～3CF	彩显 EGA/VGA 卡	彩显 EGA/VGA 卡
3D0～3DF	彩显 CGA 卡	彩显 CGA 卡
3E0～3EF	保留	保留
3F0～3F7	软驱控制卡	软驱控制卡
3F8～3FF	串行口控制卡 1	串行口控制卡 1

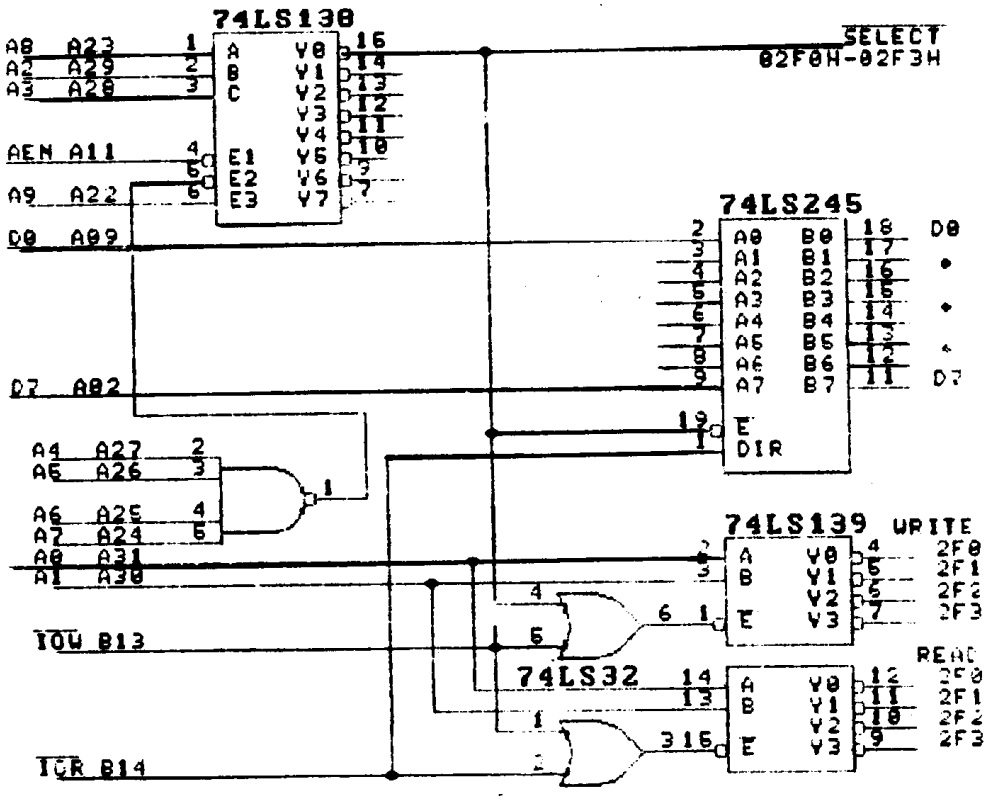


图 3.8 固定式 I/O 地址译码电路

固定式译码缺乏弹性，若想改变地址就必须改变硬件电路。图 3.9 为一开关选择式 I/O 译码电路，只要改变一组 DIP 开关之值即可改变 I/O 地址。此设计使用 74LS688 八位比较

器，比较器的一边是地址信号 A_3～A_9 和 AEN 信号，另一边则为 DIP 开关的输出。当两者相等时，即产生一选择控制信号。

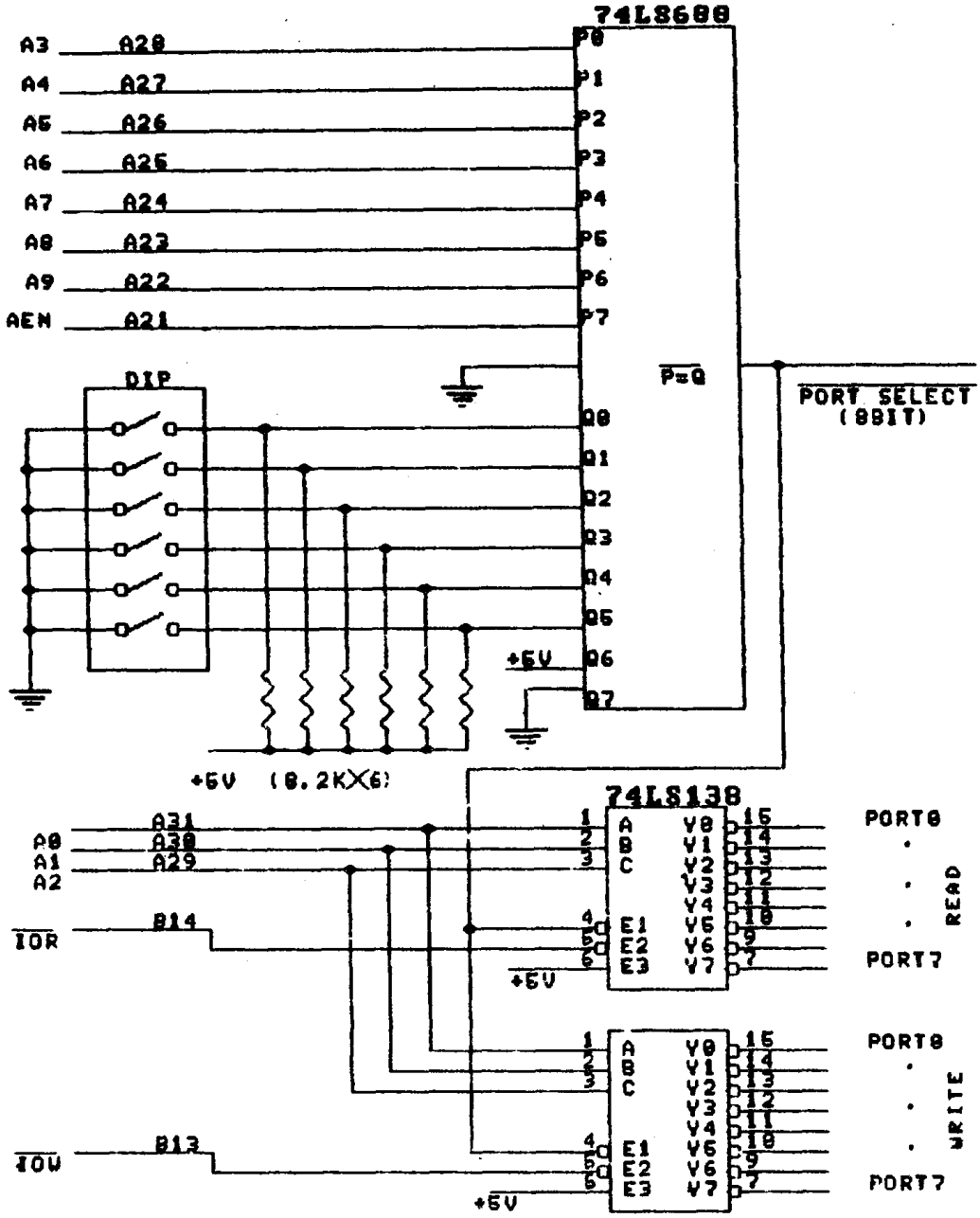


图 3.9　开关选择式 I/O 地址译码电路

二、I/O 总线驱动电路设计

PC 总线的驱动能力是很有限的，要扩展用户接口卡，总线驱动是不可少的。

在总线驱动中经常使用的芯片有三态驱动双向三态门如 74LS245 及带有三态输出的锁存器如 74LS373。一般可用 74LS244 作为单向型总线的信号增强和驱动器件，用 74LS245 作

为双向型总线(仅数据总线)的信号增强和驱动器件,信号传送的方向可以由$\overline{IOW}$和$\overline{IOR}$信号确定。74LS373 常用于地址信号的锁存和驱动。

三、I/O 总线接口电路实例

图 3.10 是总线接口的电路图,采用门电路地址译码。这里我们主要介绍三态总线收发器 74LS245 的使用方法,它是双向的,主要用于数据线。当方向控制引脚 DIR 为低电平时,数据由 B 传向 A,DIR 为高电平时,由 A 传向 B。引脚 G 则为三态门的控制端,G 为低电平时三态门打开。

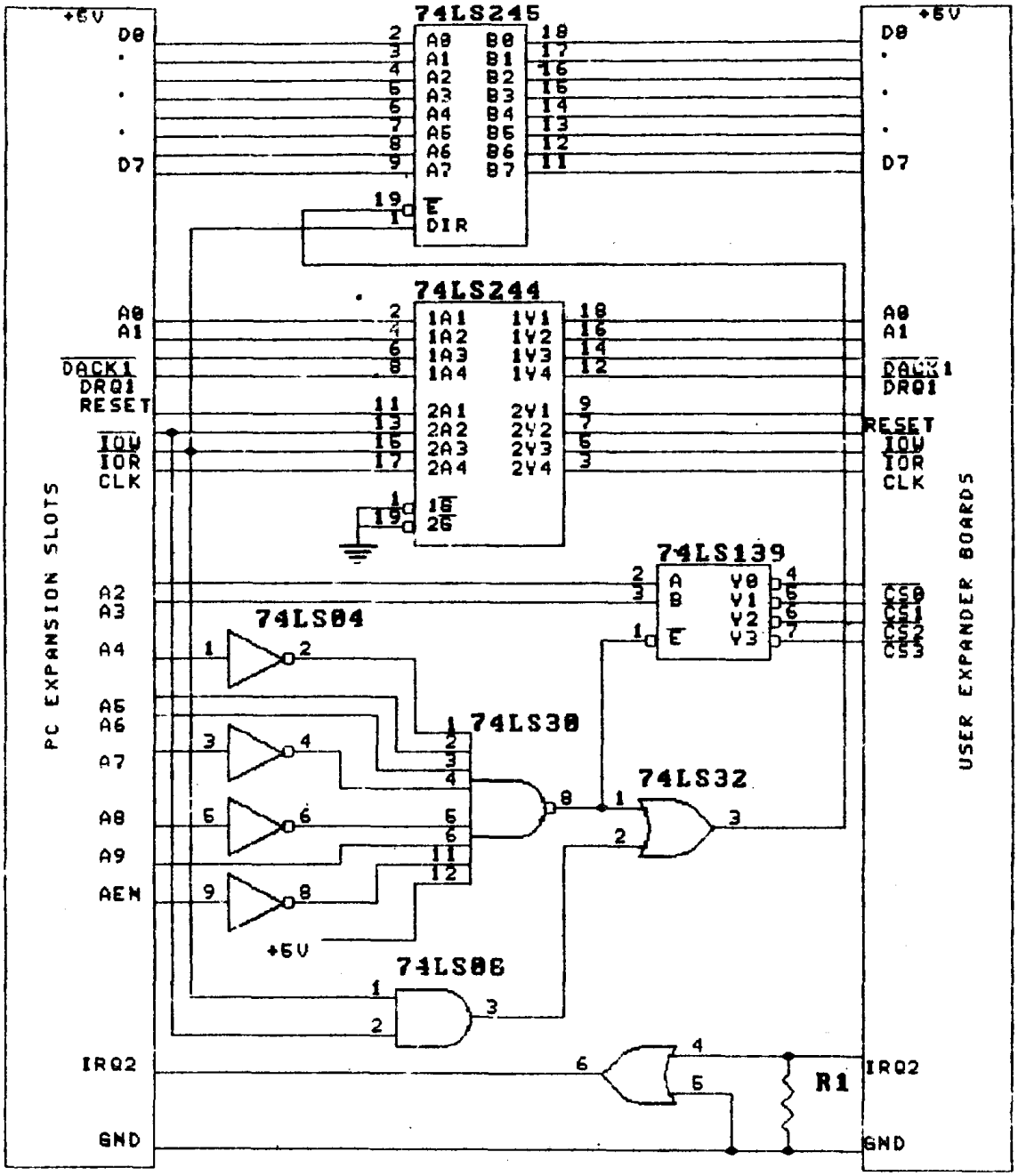


图 3.10 总线接口电路

将读控制信号$\overline{IOR}$连到 DIR 端，当读有效时，用户扩展卡一侧的数据可通过 245 输入主机，当写有效时，$\overline{IOR}$为高电平，主机一侧的数据通过 245 输出到用户扩展卡。仅当 CPU 与用户扩展卡之间进行数据交换时，三态门才打开，否则应将三态门关闭。74LS30 和 74LS139 用于地址译码，分配给用户扩展卡的 I/O 地址空间为 260H～26FH，74LS30 的输出连到 74LS32 的输入端。同时 I/O 端口的读写信号$\overline{IOR}$与$\overline{IOW}$通过 74LS08 输出到 74LS32 的另一输入端，只有当 CPU 对 260H～26FH 之间的端口地址执行输入/输出指令时，74LS32 才输出低电平信号到 245 的 G 端，这时 245 的三态门才会打开，否则总是处于高阻隔离状态。引入$\overline{IOR}$和$\overline{IOW}$信号是为了避免 CPU 访问地址为 260H～26FH 的存储器时，误将 245 的三态门打开。

由于地址线 A_0，A_1 与控制线 RESET—$\overline{DRQ_1}$是单向的，故这些信号通过总线驱动器 74LS244 与 CPU 相连即可。

上述总线接口电路为用户提供了四个片选信号$\overline{CS_0}$～$\overline{CS_3}$和常用的总线信号，本书后面的电路设计即在此基础上设计用户系统。

四、I/O 总线接口电路抗干扰问题

在进行微型计算机应用系统设计时，总线是系统设计者必须认真考虑的重要部分。除了必须保证不发生总线竞争——在同一时刻同一总线上有两个或两个以上的器件输出状态和认真考虑总线驱动能力外，还必须认真考虑电路抗干扰问题。

根据传输线理论，总线传输的信号线上存在反射、信号衰减以及噪声干扰。对时钟频率在 1～10MHz 范围内通路又被限制在一块印刷板内进行电路设计，传输线的影响一般可以忽略。但是信号用总线从一块板传到另一块板时，就有显著的传输线效应，使总线上的信号发生畸变。

1. 分布电容的影响和消除

总线信号走线过长会增加额外电容，使得容性负载随之增加。通常，电容值大于 200 皮法时，对总线信号造成的失真将导致整个系统工作不稳定。

降低分布电容的影响可采用电容卸荷法。如图 3.11 给出了电容卸荷电路。该电路由 220Ω 电阻和 47pf 电容组成，一端连到总线，另一端接到＋5V 电源。其工作原理是在静态下，电容开路，220Ω 电阻未接入总线，该电路不影响电缆的阻抗特性。当总线信号由高电平向低电平跳变时，电容间短路，将 220Ω 电路接入总线，存储在分布电容上的电荷在 200Ω 电阻上泄漏掉，从而减少负反冲作用。进入稳态后，电容开路，将电阻与总线隔离开。

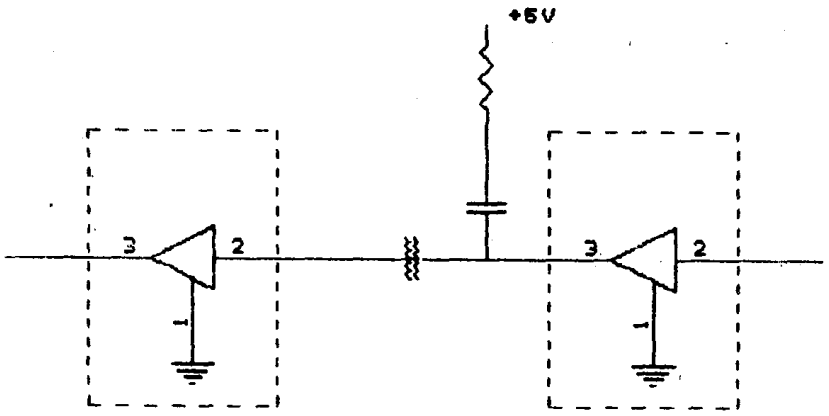


图 3.11　电容卸荷电路

2. 总线上的反射和消除

如果在总线两端没有进行阻抗匹配，信号就会在两端来回反射，使信号产生多个台阶和负反冲，造成信号畸变。解决这个问题可采用阻抗匹配法。

阻抗匹配法有两种，一种是始端阻抗匹配，一种是终端阻抗匹配。始端阻抗匹配最简单的办法是在发送驱动电路的输出与电缆的起始端之间串联一个电路而成，终端阻抗匹配最简单的办法是在接收驱动器的输入端并一个接地电阻。

3. 总线中的交叉串扰和消除

出现在板到板的连接中的噪声源之一是串扰。串扰指的是相邻的信号在这条线上所产生的噪声。相邻线中的信号通过电感和电容的作用而相互耦合，这种耦合随着相邻线长度的增加而增加，而随着两线的间隔增大而减少。采用有源端负载，是目前用于减少总线上噪声的主要技术。有源端就是通过接到公共电源的匹配阻抗来端接所有总线的方法。虽然由于传输线上的分支会在线上引起反射，但是传播到这条线末端的反射会被有源的终端负载所吸收。通常用于总线系统的驱动器都具有低、中及高阻抗三种操作方式的三态驱动器。当所有驱动器断开时，一个没有终端的总线是处在向上或向下浮动以及仍然停留在上次操作后的电位的自由状态，这就使得在驱动器将要对这条线充电时，它必须充到 V 伏。用总线传送信号时，必须把对这条线的充电延迟考虑进去。对有源终端负载来说，当所有的驱动器断开时，总线是处在逻辑阀值上，从而使得线上的电位变化不会大于 $V/2$，这样，对长线的充电延迟减少了大约一半。

4. 电源去耦

电源线常选用粗电缆进行连接，并在电缆线两端分别加上去耦电路，即用 0.01μF 的电容跨接在＋5V 和地之间以便为总线驱动器和收发器去耦。

5. 其他抗干扰措施

在设计接口卡印刷电路板时应为每块 IC 就近配置一只 0.01μF 的滤波电容，以抑制高频干扰。接口卡中的模拟地和数字地要分别相连，然后这两种地线在一点连接，以避免数字信号对模拟电路的干扰。外接输入信号时要采用屏蔽线。

3.1.3 系统时钟电路设计

以 8088，80286，80386 及 80486 为 CPU 的 PC 系列微机及兼容机均采用了 ISA 总线，总线信号保持了良好的兼容性，这为我们设计通用的微机应用系统打下了良好的基础。但是，不同机型的时钟信号频率是不一样的。80486 就要比 8088 时钟频率要高得多。为了使我们设计的微机应用系统具有通用性，最好是自己设计时钟产生电路，而不同系统总线时钟信号。

图 3.12 所示，用晶振产生频率为 4.915MHz 的时钟，经 12 级 2 分频电路 CD4040 分频，通过 DIP 开关选择构成 12 种可选频率输出，提供 1.2kHz～2.4576MHz 的用户时钟。

3.1.4 I/O 扩展卡设计步骤

当你准备设计一个 I/O 扩展卡时，一定要按步就班，不可操之过急，否则是事倍功半，如果能遵循下列步骤来进行的话，你将得到事半功倍的效果。

1. 制定扩展卡的功能规格

你的扩展卡要做什么，执行何种功能，能源消耗情况如何等，要尽可能详细记载下来，以确定自己所需要的是什么。

2. 电路方框图规划

硬件方框图之意义在于将硬件各种不同的功能模块化，使你对系统一目了然。当然你首先要对每个方框图的功能详细标示清楚，方框图与方框图之间的关系也要标示明白。

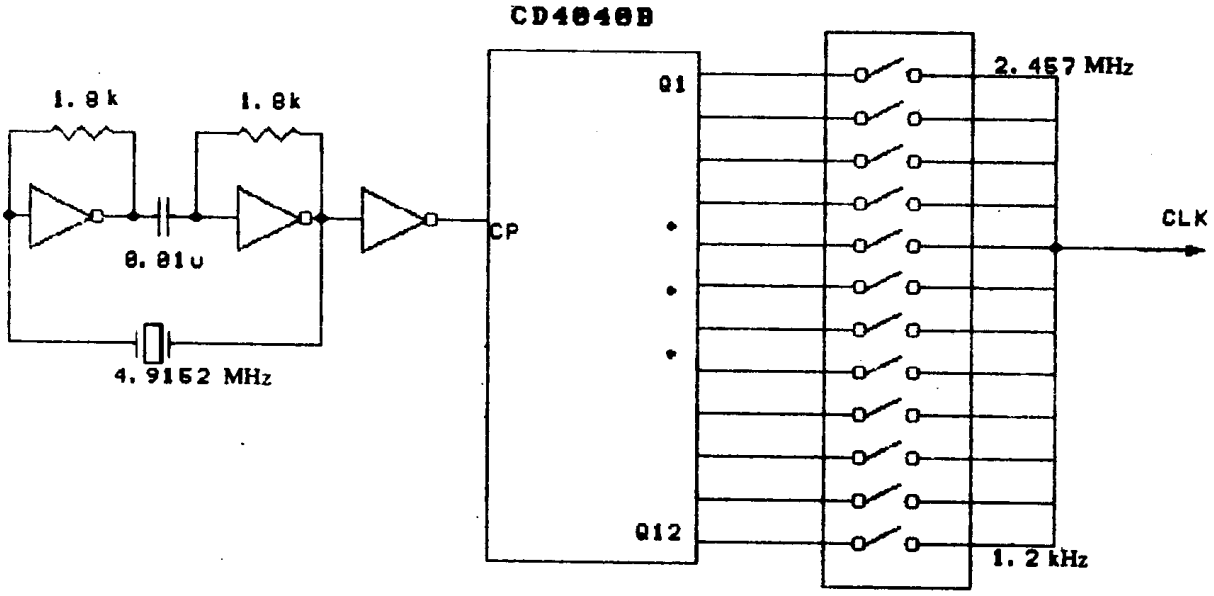


图 3.12 时钟产生电路

3. 线路设计及绘图

以类似的线路图作为设计参考，在电路方框图的指示下，设计电路图，并利用 TANGO、ORCAD 等软件绘制电路原理图。

4. 时序分析

线路图设计完成之后，接着要分析你当初的逻辑设计是否正确，使用模拟方式将各点的时序图勾画出来，仔细检查是否符合芯片要求，以及你所要求的时序。模拟方法可采用 Pspice 等模拟软件。记住！事前的分析查核会替你以后在实际制作扩展卡时省下许多时间和开支。

5. 制作印刷电路板

当线路设计及分析完毕后，下一步骤便是制作印刷电路板。通常可用 TANGO、ORCAD 等软件绘制印刷电路图，然后制板。

6. 元器件安装前检查

印刷电路板制好后，切莫迫不及待地将所有 IC 插到印刷电路板上。最好先用一个万用表检查一下＋5V 电源与接地之间的电阻值，如果此电阻值比较小(非无穷大)，则表示可能某个地方的电路有短路现象。仔细检查并改正，至此项测试通过，方可将用户扩展卡插到扩展槽内，再检查应有的电源电压的接脚上是否都有了正确的电压，不该有电压的地方是否就无电压呢？

7. 元器件安装

经过上一步检查，如果一切正常，即可安装元器件。插入 IC 时，首先要确定其连接脚 1

是否对准插座上的接脚 1 插孔。

8. 系统调试

扩展卡设计的最后一步就是系统的调试。在调试时，元器件要逐个插入电路板中，接着进行调试。首先调试那些独立的或相对孤立的部分。一次把元器件全插上去，会互相干扰，一旦出现问题，造成的损失就更大了。

调试可采用如下方法：(1)根据调试硬件部分的功能编写相应的调试软件；(2)用万用表，示波器等观察电路各点的时序，检查是否符合要求；(3)用 IN 和 OUT 指令对输入和输出端口进行数据读写，判断输入与输出结果是否正确。

3.2 中断接口扩展卡的设计

3.2.1 中断控制器 8259 的引脚与功能

8259 是 28 个引脚封装的双列直插式芯片，只需单一的＋5V 电源，不需时钟输入。芯片的引脚排列如图 3.13，各引脚的功能如下：

IR_0～IR_7：8 个中断请求输入端，对于边沿触发方式，IR 由低变高就产生中断请求，并维持高电平直至被响应；对于电平触发方式，IR 仅是一个高电平脉冲。

INT：中断请求输出端，接 CPU 中断请求输入。

$\overline{INTA}$：中断响应输入端，用于接受 CPU 的中断响应脉冲$\overline{INTA}$，将中断向量送到数据总线。

$\overline{CS}$：片选，为低时该片被选中，才可对它进行读写。

$\overline{WR}$：写信号，为低时允许 CPU 将初始化命令字和操作命令字分别写入内部相应的寄存器。

$\overline{RD}$：读信号，为低时 CPU 可对芯片内部 IRR、ISR、IMR 内容以及中断向量进行读出。

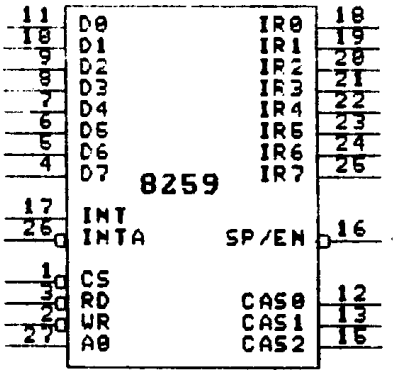


图 3.13 8259 芯片引脚排列

A_0：地址线，与$\overline{WR}$、$\overline{RD}$信号配合使用，用选择不同的命令寄存器写或不同的状态寄存器读。

D_0～D_7：双总数据总线，用于传送命令、状态和中断向量。

SP/$\overline{EN}$：从片/缓冲控制线，是一条双功能线，在缓冲器方式下，该端为输出，控制缓冲器使能。在其他方式下为输入，用于多片级联时，主从片的区别，当 sp＝1 时为主片，当 sp＝0 时为从片。

CAS_0～CAS_2：级联线，在多片级联时，这些端并联相连，主片作为输出，从片作为输入，主片通过这些线发出从片中断请求识别码，当从片识别码和它相符时，该从片中断被响应。

3.2.2 中断接口扩展实例

例 3.1 8259 中断接口扩充实例

可屏蔽硬中断请求线的扩充，实际上就是增加 8259 中断控制器的数目。由于 PC 机 62 线插槽上只引出了中断申请线 IRQ，而没有引出 8259 的级联总线 CAS_0～CAS_2 和 CPU 的中

断响应信号$\overline{INTA}$，故不能采用多片级联的方法进行扩充。也就是说，不能采用向量中断，只能采用查询中断方法。具体作法是将附加的 8259A 的中断请求线 INT 连接到 62 线插槽的某一 IRQ_i 上（该线应为系统暂不用的，如 IRQ_2），由它引到主板上的 8259 的中断输入端 IR_i。为了对附加的 8259 写入命令字 ICW 和 OCW，需给它分配两个口地址 $PORT_0$（$A_0=0$），$PORT_1$（$A_0=1$）。如图 3.14 所示。

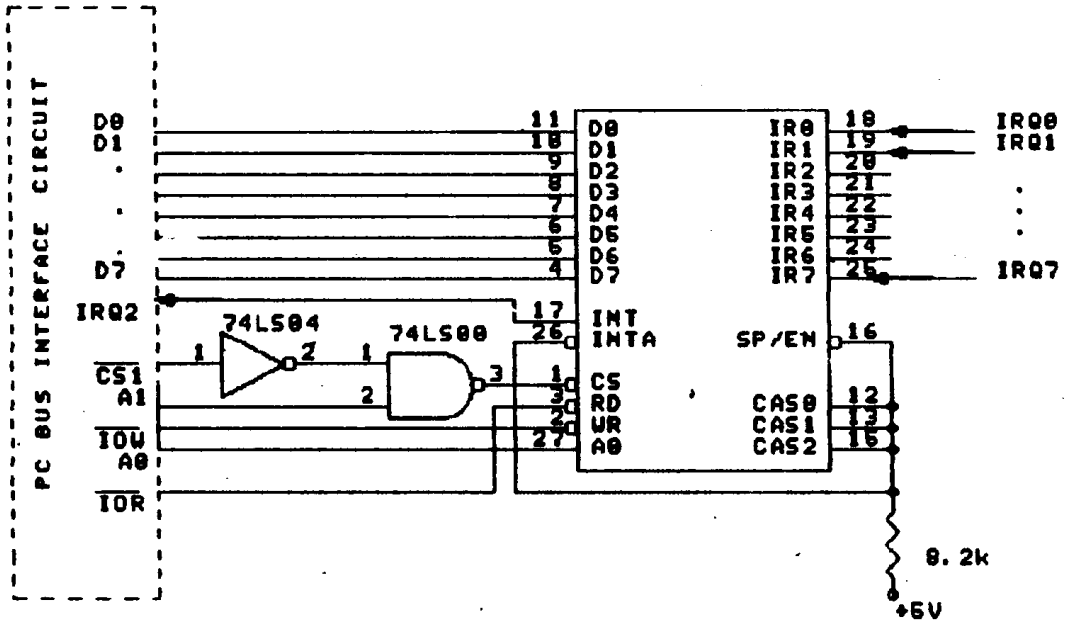


图 3.14　中断请求扩充

当附加的 8259 初始化后，若扩充的中断系统有中断源请求中断且允许响应，则附加的 8259 向系统板上的主 8259 发出中断请求信号 INT。主 8259 接收后，向 CPU 发出 INT 信号。这时若 CPU 响应，便发出$\overline{INTA}$信号，从而主 8259 获得中断类型号，以此转入相应的中断服务程序（至此完成一级中断处理，采用向量中断方式）。在此中断服务程序的开头部分，安排一条第二位（$D_2=P$）$=1$ 的 OCW_3 命令使附加 8259 工作在查询方式。在这种方式下，附加的 8259 不要求从 CPU 收到中断响应信号$\overline{INTA}$，只需 CPU 发读命令，便可读取它的查询字。根据查询字的 W_2，W_1，W_0 编码，转到相应的中断服务程序中去（至此完成二级中断处理，是采用查询中断方式）。当中断源中断服务程序结束后，先返回系统的主 8259 中断程序，再从该程序返回到主程序。程序清单如下：

```
;     exam31.asm

stack     segment para stack ' stack'
          db   150 dup(0)
stack     ends
data      segment
mesg0     db 0ah,0dh,"int0 is serving $"
mesg1     db 0ah,0dh,"int1 is serving $"
mesg2     db 0ah,0dh,"int2 is serving $"
mesg3     db 0ah,0dh,"int3 is serving $"
```

```
mesg4   db 0ah,0dh,"int4 is serving $"
mesg5   db 0ah,0dh,"int5 is serving $"
mesg6   db 0ah,0dh,"int6 is serving $"
mesg7   db 0ah,0dh,"int7 is serving $"
port0   equ 266h
port1   equ 267h
data    ends
code    segment
        assume cs:code,ds:data,ss:stack
init    proc far
start:  push ds
        mov ax,0
        push ax
        mov ax,data
        mov ds,ax
        mov ax,stack
        mov ss,ax
        mov dx,port0            ;ICW1 口地址
        mov al,13h              ;边沿触发,单片使用,需要 ICW4
        out dx,al
        mov dx,port1            ;ICW2 口地址
        mov al,00h              ;ICW2 的内容,可以任意,因为不是采用向量中断
        out dx,al
        mov al,05h              ;ICW4=一般嵌套,非缓冲方式,主控,8088
        out dx,al
        mov al,00h              ;OCW1=开放所有中断请求
        out dx,al
        push ds
        mov ax,seg intr1        ;中断服务程序入口段地址
        mov ds,ax
        mov dx,offset intr1     ;中断服务程序入口偏移地址
        mov ah,25h
        mov al,0ah              ;中断类型号
        int 21h
        pop ds
again0: sti
        jmp again0
        ret
init    endp
intr1   proc far
        push ax
        push dx
        push cx
```

```
        mov dx,port0          ;OCW3 口地址
        mov al,0ch            ;OCW3=查询方式
        out dx,al
        nop
        in al,dx              ;读 8259A 查询字
        and al,07h
        cmp al,07h            ;是 7 号中断源请求中断吗?
        jz num7               ;是,则转向 7 号中断服务程序
        cmp al,06h            ;是 6 号中断源请求中断吗?
        jz num6               ;是,则转向 6 号中断服务程序
        cmp al,05h            ;是 5 号中断源请求中断吗?
        jz num5               ;是,则转向 5 号中断服务程序
        cmp al,04h            ;是 4 号中断源请求中断吗?
        jz num4               ;是,则转向 4 号中断服务程序
        cmp al,03h            ;是 3 号中断源请求中断吗?
        jz num3               ;是,则转向 3 号中断服务程序
        cmp al,02h            ;是 2 号中断源请求中断吗?
        jz num2               ;是,则转向 2 号中断服务程序
        cmp al,01h            ;是 1 号中断源请求中断吗?
        jz num1               ;是,则转向 1 号中断服务程序
        cmp al,00h            ;是 0 号中断源请求中断吗?
        jz num0               ;是,则转向 0 号中断服务程序
eoi:    mov dx,port0          ;附加 8259A 的 OCW2 的口地址
        mov al,cl             ;指定中断结束,刚服务过的中断复位
        out dx,al
        mov al,62h            ;OCW2=指定中断结束,ISR2 复位
        out 20h,al            ;主 8259A 的 OCW2 的口地址
        pop ax
        pop dx
        pop cx
        iret
num7:   nop                   ;7 号中断服务程序
        mov ah,09h
        mov dx,offset mesg7
        int 21h
        mov cl,67h
        jmp eoi
num6:   nop                   ;6 号中断服务程序
        mov ah,9
        mov dx,offset mesg6
        int 21h
        mov cl,66h
        jmp eoi
```

```
num5:   nop                     ;5 号中断服务程序
        mov ah,9
        mov dx,offset mesg5
        int 21h
        mov cl,65h
        jmp eoi
num4:   nop                     ;4 号中断服务程序
        mov ah,9
        mov dx,offset mesg4
        int 21h
        mov cl,64h
        jmp eoi
num3:   nop                     ;3 号中断服务程序
        mov ah,09h
        mov dx,offset mesg3
        int 21h
        mov cl,63h
        jmp eoi
num2:   nop                     ;2 号中断服务程序
        mov ah,9
        mov dx,offset mesg2
        int 21h
        mov cl,62h
        jmp eoi
num1:   nop                     ;1 号中断服务程序
        mov ah,9
        mov dx,offset mesg1
        int 21h
        mov cl,61h
        jmp eoi
num0:   nop                     ;0 号中断服务程序
        mov ah,9
        mov dx,offset mesg0
        int 21h
        mov cl,60h
        jmp eoi
intr1   endp
code    ends
        end start
```

3.2.3 中断接口实验

实验 3.1 中断控制器 8259 的工作原理和应用

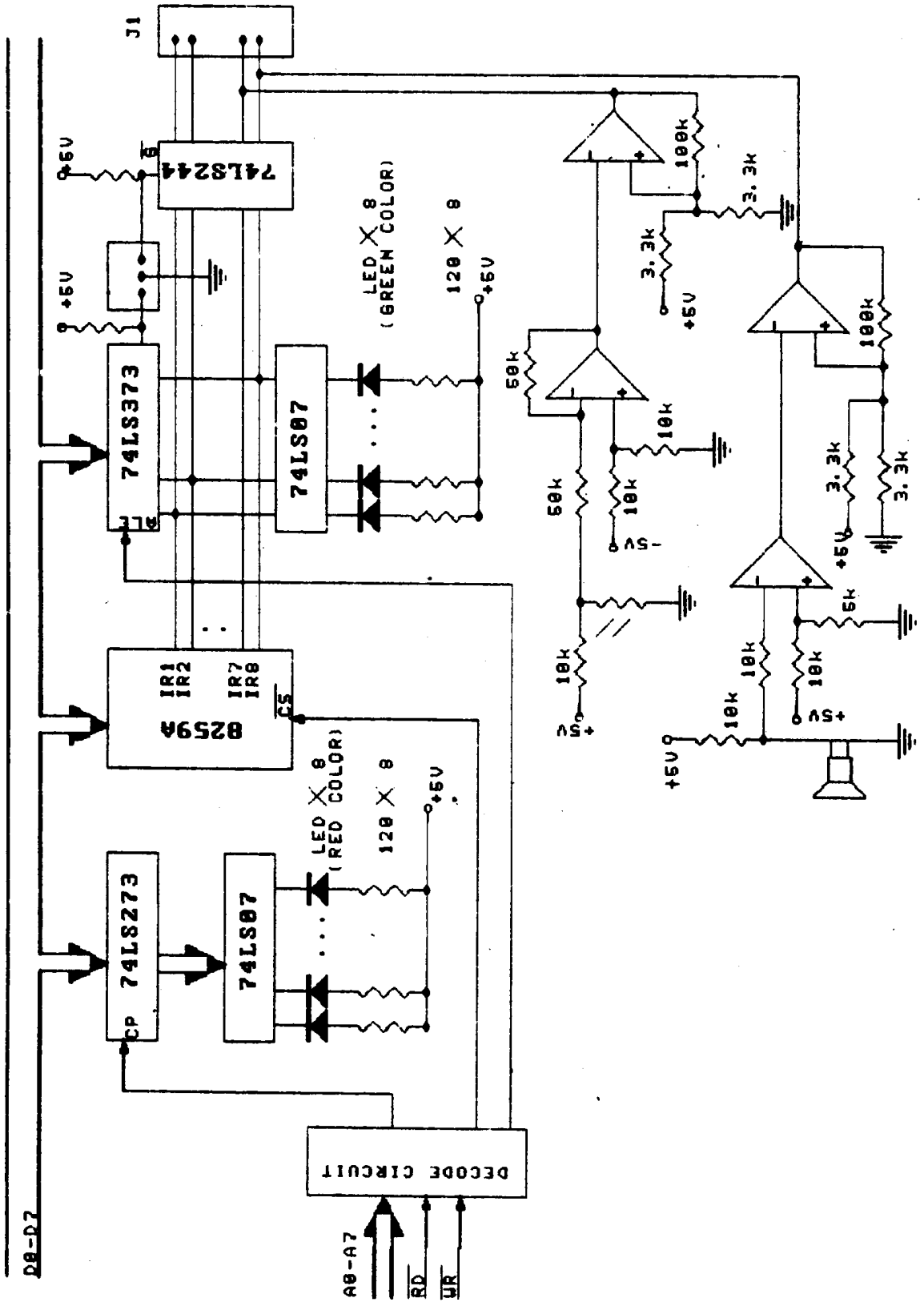


图 3.15 8259 中断实验原理框图

一、实验原理

本实验原理如图 3.15 所示，它由地址译码，可编程中断控制器 8259，模拟中断源电路及中断状态，显示电路四个部分组成，具有模拟中断请求和显示中断程序状态的功能。中断服务的控制过程由软件实现。

本实验的中断源可由两种方法获得：(1)通过 74LS373 输出口由软件产生中断信号；(2)从外部端口引入中断信号，其中包括声音检测电路和光电转换电路。为显示中断请求的需要，还特意设置了一个中断请求状态显示电路。

中断状态显示电路由一个 8D 触发器 74LS273 输出口通过反相器驱动 8 个发光二极管。发光二极管发亮表示通过 8259 后的 CPU 正在响应一个中断并正在对相应的中断请求服务。

二、实验要求

1. 根据图 3.15 所示原理框图设计线路图，确定 8259 的端口地址。

2. 走马灯实验

(1)用跳线将 74LS244 缓冲器的 1Q(1 脚)和 1EQ(19 脚)接高电平，成关闭状态。

(2)参考图 3.16 所示的流程框图，编制走马灯(8 个发光二极管循环点亮)的中断控制程序、初始程序和中断服务程序。注意：八个中断源可共用一个中断服务子程序。

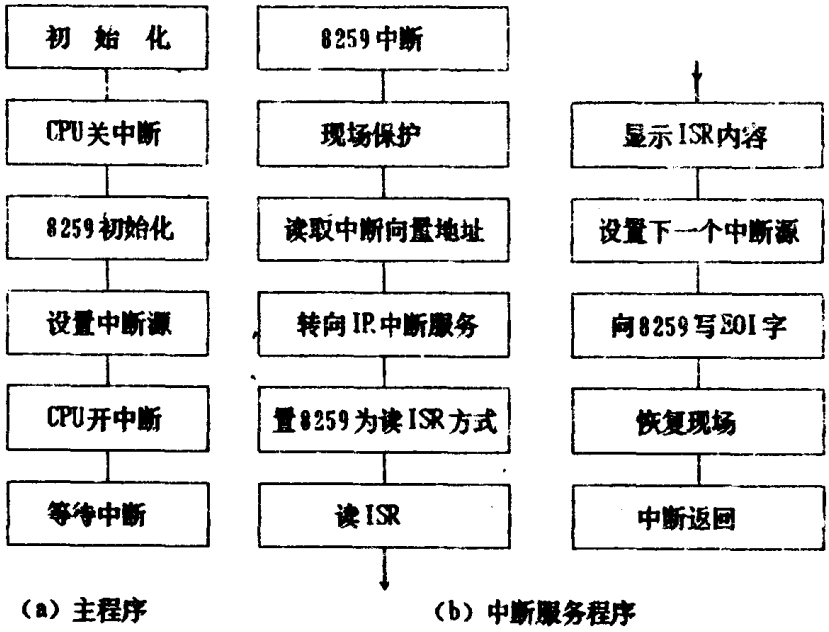


图 3.16 走马灯程序流程图

(3)运行中断控制程序，并观察 8259 的中断请求端口的状态(通过绿色发光二极管)和 8259 的中断服务寄存器的状态(通过红色发光二管)，分析 8259 的工作状态。

(4)向 8259 写 OCW_1 屏蔽一个或几个 IR 端口，再重复第 3 步。

3. 遮光实验，拍手实验

(1)将 74LS244 缓冲器用跳线选通(脚 1 和脚 19 接低电平)。

(2)参考走马灯中断控制程序,编写遮光实验和拍手实验中断控制程序。

(3)运行中断控制程序,并用手遮住光敏电阻的光源,观察遮光实验的发光二极管显示情况,并分析其原因。

(4)运行中断控制程序,并拍打双手,观察拍手实验的发光二极管显示情况,分析其原因。

3.3 定时/计数接口扩展卡的设计

3.3.1 定时/计数芯片 8253-5 的引脚与功能

8253-5 是一种 24 脚封装的双列直插式芯片,其引脚和功能结构示意图如图 3.17。各引脚的定义如下。

D_0~D_7:数据线

A_0,A_1:地址线,用于选择 3 个计数器中的一个及选择控制字寄存器。

$\overline{RD}$:读控制信号,低电平有效。

$\overline{WR}$:写控制信号,低电平有效。

$\overline{CS}$:片选端,低电平有效。

$CLK_{0\sim2}$:计数器 0#、1#、2#的时钟输入端。

$GATE_{0\sim2}$:计数器 0#、1#、2#的门控制脉冲输入端。

$OUT_{0\sim2}$:计数器 0#、1#、2#的输出端。

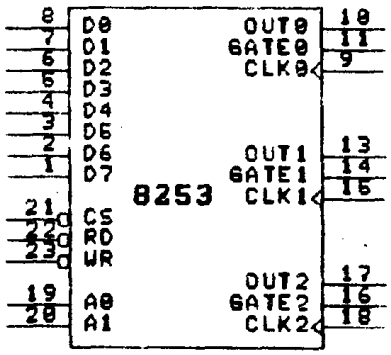


图 3.17 8253 芯片的引脚排列

8254 是 8253 的改进型,因此它的操作方式以及引脚与 8253 完全相同,其改进主要反映在两个方面:(1)8254 的计数频率更高,可高达 10MHz。(2)8254 多了一个读回命令,8254 中每个计数器都有一个状态字可由读回命令令其锁存,然后由 CPU 读取,以便了解 8254 的工作状态。

3.3.2 定时/计数接口扩展实例

例 3.2 用 8254 测量频率

采用计数器/定时器接口电路测量转速或频率、周期及脉宽等时间量是非常方便的。采用 8254 测量转速或频率的电路图如图 3.18 所示。令 8254/8253 芯片的通道 0 和 1 都工作于方式 3,并将它们级联起来,通道 0 和 1 的计数初值都设为 2000,于是通道 1 的输出脚 OUT_1 输出的方波频率为:

$$f=\frac{2\times10^6}{2000\times2000}(Hz)=0.5Hz$$

这一频率的方波接到通道 2 的 $GATE_2$。由于 f=0.5Hz,所以方波周期 T=2s,其高电平为 1s,恰好获得 1s 的门控时间。通道 2 工作于方式 0,初值设 9999。初值与被测信号计数之差即为被测频率。端口地址为 260~263H,程序清单如下:

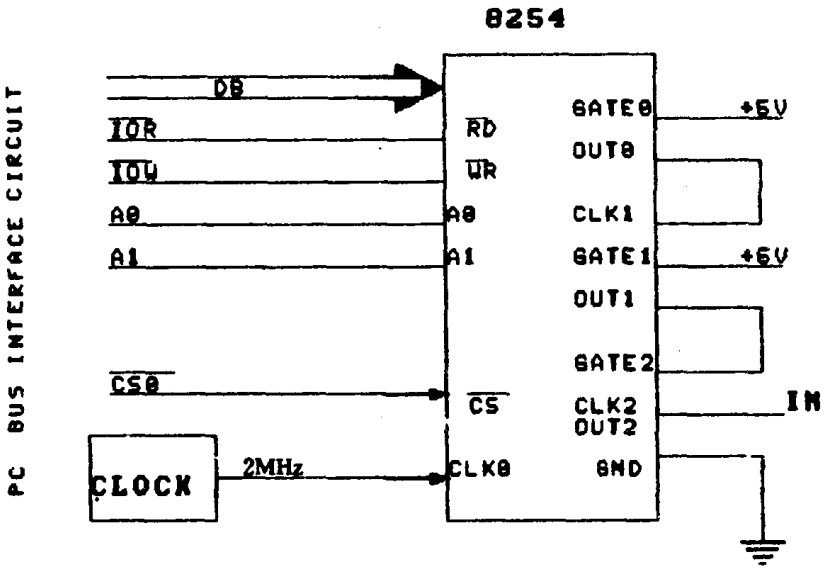


图 3.18　8254 测频率电路图

```
;       exam32.asm

data    segment
buffer  db 1000 dup(0)
messg   db ' data overflow' ,0ah,0dh,' $'
data    ends
stack   segment para stack ' stack'
        db 256 dup(?)
stack   ends
code    segment
        assume cs:code,ds:data,ss:stack,es:data
main    proc far
start:  push ds
        xor ax,ax
        push ax
        mov ax,data
        mov ds,ax
        mov es,ax
        mov ax,stack
        mov ss,ax
        lea di,buffer
        mov al,36h                      ;0 通道方式 3,二进制计数
        mov dx,263h
        out dx,al
        mov al,76h                      ;1 通道方式 3,二进制计数
        out dx,al
        mov al,0d0h                     ;初值低八位
```

```
        mov dx,260h
        out dx,al                    ;写入计数器 0
        inc dx
        out dx,al                    ;写入计数器 1
        mov al,07h                   ;初值高八位
        out dx,al                    ;写入计数器 1
        dec dx
        out dx,al                    ;写入计数器 0
cycle:  mov al,0b1h                  ;二通道方式,BCD 码计数
        mov dx,263h
        out dx,al
        mov al,99h                   ;初值
        dec dx
        out dx,al                    ;写入低八位
        out dx,al                    ;写入高八位
stat:   mov al,0e4h                  ;写 1 通道读回命令
        mov dx,263h
        out dx,al
        dec dx
        dec dx
        in al,dx                     ;读入 1 通道状态
        rcl al,1
        jc stat                      ;未测定,等到低电平
        mov al,0c8h                  ;写 2 通道读回命令
        inc dx
        inc dx
        out dx ,al
        dec dx
        in al,dx                     ;读入 2 通道状态
        rcl al,1
        jc ovflw                     ;超量程(计数值>9999)
        in al,dx                     ;读低位计数
        call adj                     ;作减法
        inc al                       ;第一个 CLK 脉冲没有计数
        stosb                        ;保存计数低位
        in al,dx                     ;读高位计数值
        call adj                     ;作减法
        stosb                        ;保存计数高位
        jmp cycle
ovflw:  in al,dx
        in al,dx
        mov dx,offset messg
        mov ah,09h
```

```
        int 21h
        mov ah,01h
        int 21h
        cmp al,' q'
        jz exito
        cmp al,' Q'
        jnz cycle
exito:  ret
main    endp
adj     proc near
        mov ch,al
        mov al,99h
        sub al,ch
        ret
adj     endp
code    ends
        end start
```

3.3.3 定时/计数接口实验

实验 3.2 电子计数器

一、实验原理：

电子计数器的多周期同步测量的原理如图 3.19(a)所示。f_x 为输入信号频率，f_0 为时钟脉冲的频率，F_x 和 F_0 两个计数器在同一闸门时间 T 内分别对 f_x 和 f_0 进行计数，计数器 F_x 的计数值，$N_x = f_x \cdot T$，计数器 F_0 的计数值 $N_0 = f_0 \cdot T$，由此可得被测频率为

$$f_x = (N_x/N_0) \cdot f_0$$

同步电路(D 触发器)的作用在于使开门信号与被测信号 f_x 同步，实现同步开门使开门时间 T 准确地等于被测信号周期的整数倍，故上式中的计数值 N_x 没有±1 字误差。但开门信号与时钟信号 f_0 不同步，计数值 N_0 有±1 字误差，由于时钟频率 f_0 很高，$N_0 \gg 1$，故 N_0 的±1 的相对误差($\pm 1/N_0$)小，且该误差与被测频率 f_x 无关，因此在整个测频范围内，多周期(即 N_x 个周期)测量为等精度的测量。

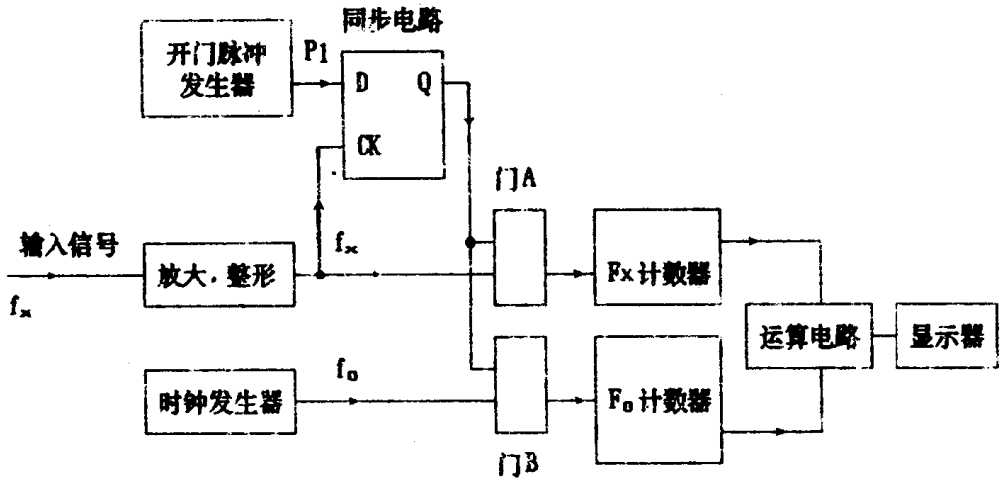
根据图 3.19(a)的原理图，设计出由 8253 计数器组成的电路原理框图如图 3.19(b)，各功能部件组成如下：

F_0 计数器：由 8253 的计数器 0 和计数器 1 级联而成的一个 8 位 BCD 码计数器。

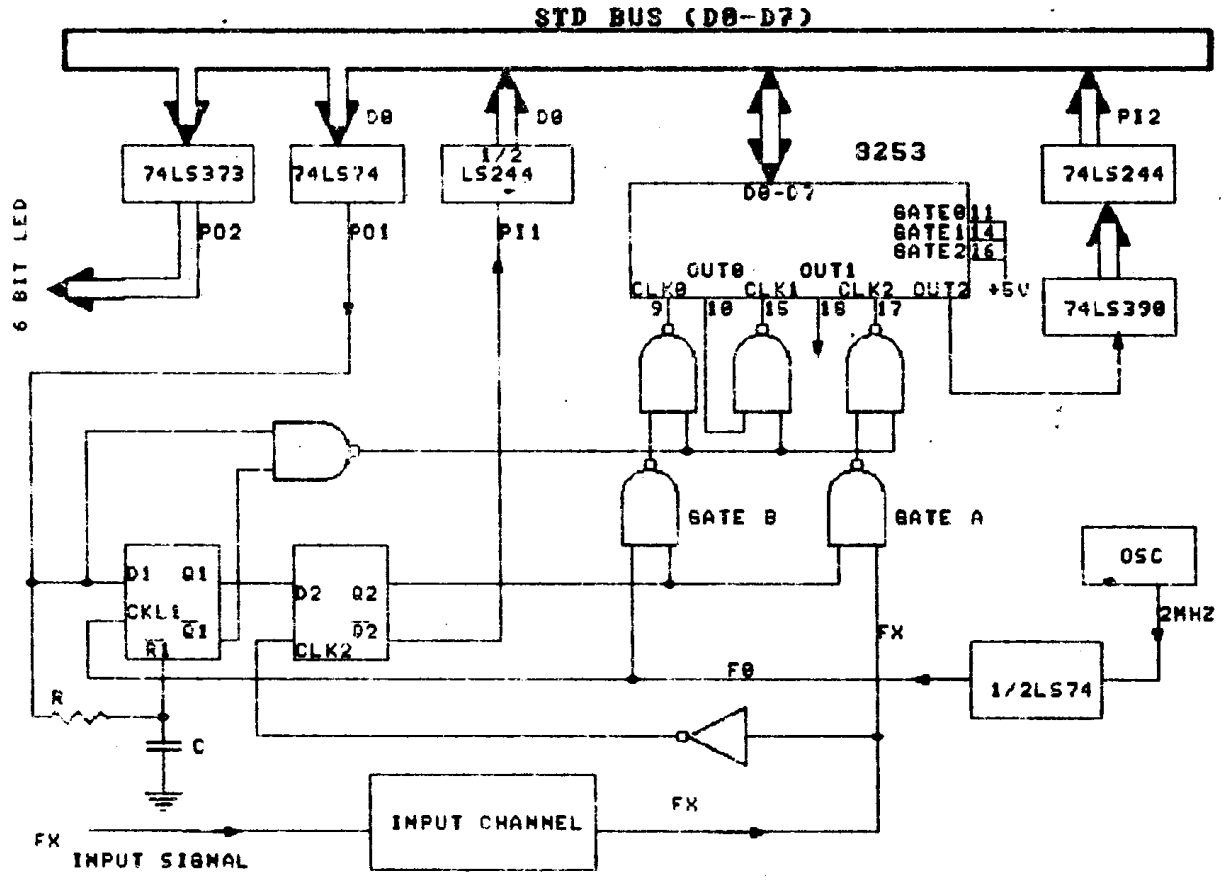
F_x 计数器：由 8253 的计数器 2 和 74LS390 级联而成一个 6 位 BCD 码计数器。

时钟发生器：1MHz 的时钟信号由 2MHz 晶体振荡器经二分频后产生。

同步闸门：同步闸门由双 D 触发器构成。开门信号由软件产生，经 PO_1 口输出至 D_1 端，当软件发出开门信号后，D 触发器 1 的翻转产生一个开门前的置数脉冲(经与非门 G 输出)，完成 8253 内的三个计数器从初值锁存器往计数器打入计数初值的置数脉冲作用。D 触发器 2 完成对 f_x 信号的同步开门作用。经同步后的实际闸门开启或关闭状态，CPU 可由 PI_1 输入口检测。



(a)原理框图



(b)电路逻辑框图

图 3.19　电子计数器原理框图

本实验板上除 8253 的计数器 0、计数器 1、计数器 2 和控制寄存器 4 个端口外，还提供了 4 个端口 PI_1、PO_1、PI_2、PO_2，其功能如下：

PI_1：输入口，从 D_0 位读入同步后的闸门启闭状态(0 为开启，1 为关闭)。

PO_1：输出口，从 D_0 位发出闸门信号(0 为关闭，1 为开启)。

PI_2：输入口，74LS390 的计数值输入口，2 位 BCD 码。

PO_2：输出口，测量结果显示输出口，其中 D_3～D_0 为 BCD 的数字码，D_6～D_4 为位选码。

二、实验要求

1. 根据图 3.19 所示的原理框图设计线路图，确定 8253 的四个端口地址和 PI_1、PO_1、PI_2、PO_2 四端口的地址。

2. 根据图 3.20 所示的程序流程图设计测频程序。

3. 将信号源输出接至计数模板上测频输入 f_x 端，并将 6 位 LED 显示板与计数模板相连。

4. 在 PC 机上运行程序，并记录实验数据，改变输入频率 f_x，并记录测量结果，进行结果分析。

三、提示

1. 8253 的初始化

根据实验的要求，8253 计数器 0、计数器 2 设置为工作方式 2，而计数器 1 设置为方式 0。为省去 f_x 运算结果的数值转换，8253 采用 BCD 计数，各计数器的初值可写入 9999H。

2. 计数闸门的开启与闭合

计数闸门由输出口 PO_1 的 D_0 位控制，"1"为开闸门，而实际计数闸门由 f_x 同步得到，它可由 PI_1 的 D_0 检测到，"0"为同步闸门开启。闸门时间由软件延时得到(调用子程序 D100ms 可得 100ms 延时)。

编写控制闸门的开、关软件应注意：在软件发出开闸门信号后，应等待同步闸门开启，若等待时间超过 1s，则认为 $f_x=0$；在软件发出关闸门信号后，也应等待同步闸门关闭，然后读数，若等待超过 1s，亦认为 $f_x=0$。

3. 计数值的读取

读数操作包括对 8253 读数及对 74LS390 读数，74L390 为加法计数，直接读端口 PI_2 即可。8253 的每个 16 位计数器的计数结果需两次读出，高位和低位字节的读出方式由方式控制字中 RL_1，RL_2 决定，有两种读出方式：直接读数和闩锁读数。由于 8253 为减法计数器，欲得到各计数值的真值须由初值 9999 减去读数值。

4. 计数频率

由前述多周期同步测量原理，被测信号频率 $f_x=(N_x/N_0)\cdot f_0$，需通过计算获得。本模块的时钟频率 f_0 已设定为 1MHz，只需将 N_x 后添上 6 个零再除以 N_0 即可获得 f_x 值。

5. 显示 f_x 值

f_x 的等精度测量取 6 位有效数字显示。由于数据采用定点数运算，因此需注意小数点位置的显示(单位为 Hz)。另外，为表示出测量闸门，设置一标志位 FLASH，每测量一次取一次反，当 FLASH="1"时，小数点亮，这样，便可显示出一亮一灭的小数点。

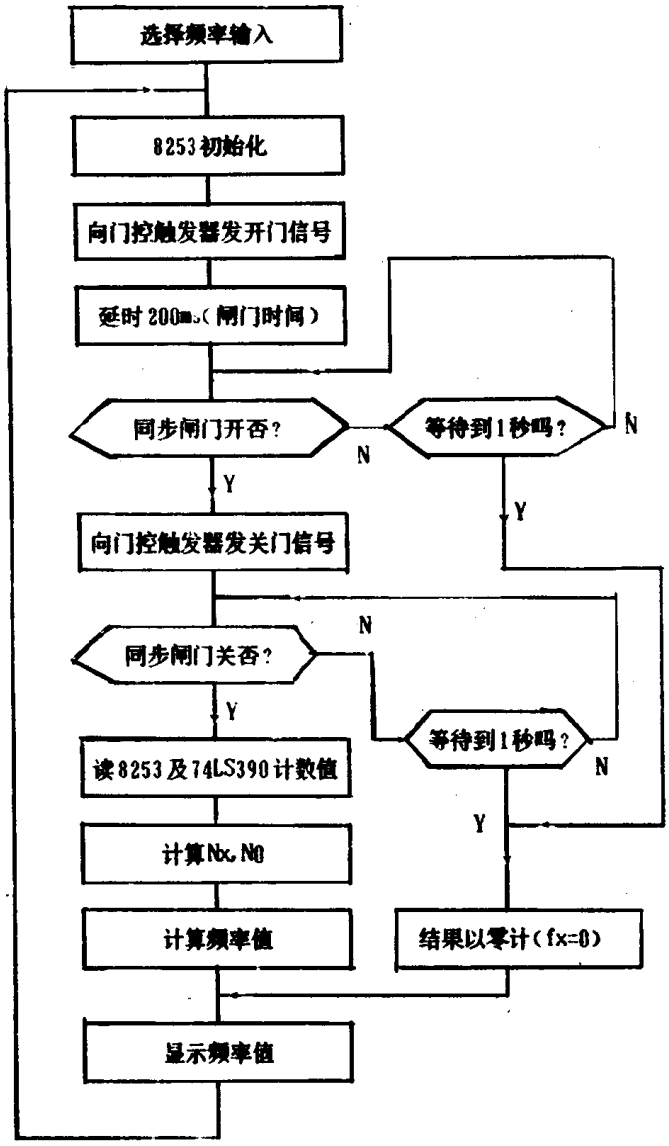


图 3.20　测频程序流程图

3.4　键盘/显示 接口扩展卡设计

3.4.1　键盘/显示接口芯片 8279 的引脚与功能

8279 是一个 40 脚封装的双列直插式芯片。图 3.21 是其引脚和功能示意图。

$DB_0 \sim DB_7$：数据线

$\overline{RD}$：读控制线，低电平有效。

$\overline{WR}$：写控制线，低电平有效。

$\overline{CS}$：片选端，低电平有效。

$C/\overline{D}(A_0)$：读写控制命令/数据线。$C/\overline{D}=1$，可对 8279 读写控制状态写命令；$C/\overline{D}=0$，可对 8279 读写数据。

CLK：内部定时时钟输入信号。

RESET：复位信号，高电平有效。

IRQ：中断请求输出信号，当 8279 查到有键按下时能自动把按键信息输入 8279 内部的键盘 RAM，并使 IRQ 置位，发中断请求。若 CPU 响应中断请求，便可以读入键盘 RAM 中的键信息并使 IRQ 复位。

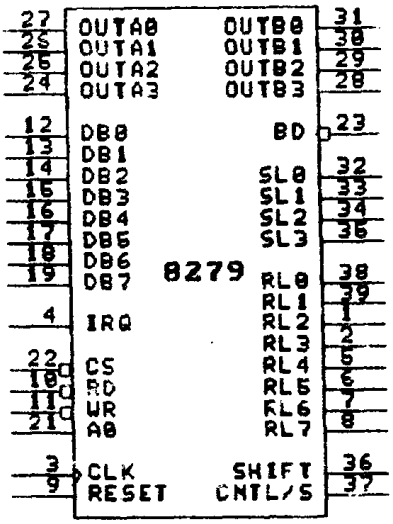


图 3.21　8279 芯片的引脚排列

$SL_{0\sim3}$：扫描输出线。如按编码扫描方式工作，可以提供 4 根扫描控制线；如果再附加译码电路按译码扫描方式工作，最多可提供 16 根扫描控制线。

$RL_{0\sim7}$：按键或传感器矩阵输入线，它们与 $SL_{0\sim3}$ 一起组成查键扫描矩阵。

SHIFT：输入控制线。用来区分上档/下档键。

CNTL/S：输入控制线。在一般输入方式，CNTL 作为控制信息可以同时读入键盘 RAM。在选通脉冲工作方式，该输入线作为选通脉冲(STB)输入端。当该线输入一正选通脉冲时，即可把键状态信息(作为传感器接点状态)锁存入键盘 RAM。

$OUTA_{0\sim3}$，$OUTB_{0\sim3}$：二组显示控制输出线。用它们控制七段 LED 显示的亮与灭。

$\overline{BD}$：显示熄灭控制输出线。它输出一个控制信号通过硬件使七段 LED 显示熄灭。

3.4.2　键盘/显示接口实例

例 3.3　键盘/显示接口

硬件电路如图 3.22 所示，键盘由 8×8 矩阵组成，8 位 LED 显示采用共阴极结构。8279 的扫描输出线 $SL_{0\sim3}$，经译码电路(74LS138)和驱动电路(7407)产生 8 个扫描控制信息 $R_0\sim R_7$，作为键入行扫描控制与七段 LED 显示位码控制。8279 的 $OUTA_{0\sim3}$ 与 $OUTB_{0\sim3}$ 的输出线经 8 个 PNP 三极管驱动产生 LED 显示器段码控制信号 $C_0\sim C_7$。8279 端口地址为 266H($C/\overline{D}=A_0=0$)和 267H($C/\overline{D}=A_0=1$)。

8279 键盘、显示器编程时，首先需对 8279 初始化，指定键盘、显示工作方式。编程中较多使用的命令是写显示 RAM 和读键值，由于 8279 能实现自动扫描，因此虽然显示方式是动态扫描方式，但软件操作上却似静态方式，因而编程很简单，读键值时首先查询 8279 工作状态，8279 内部有 8 个字节的键盘 RAM，存放着按键的个数，8279 的键值需读数据端口，

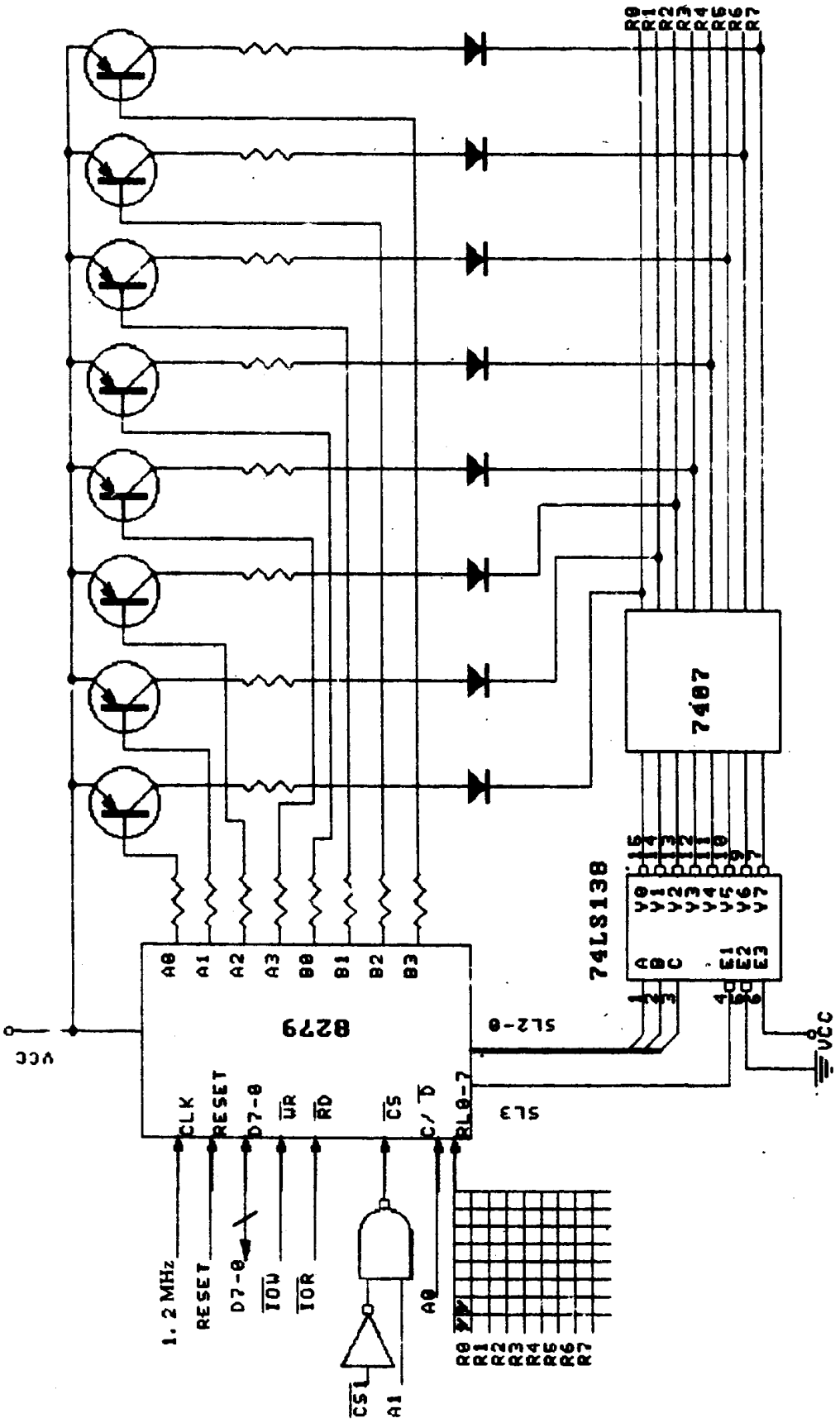
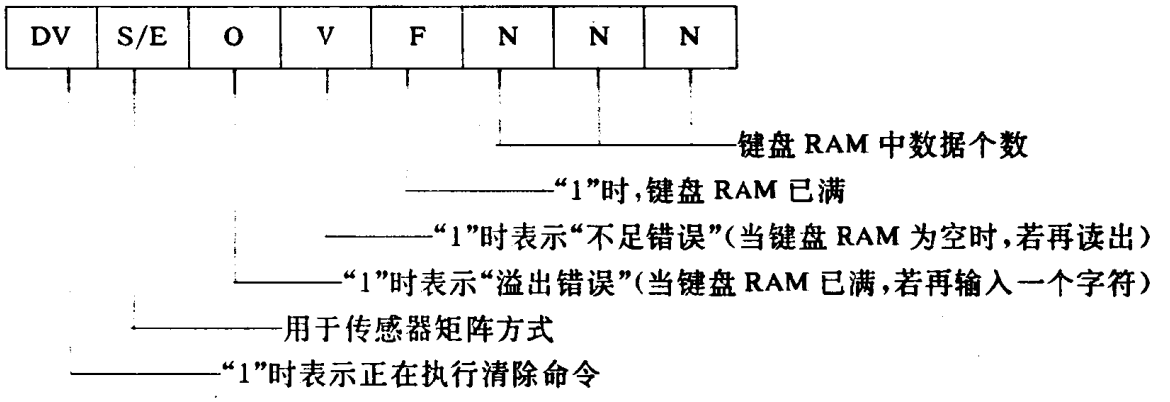


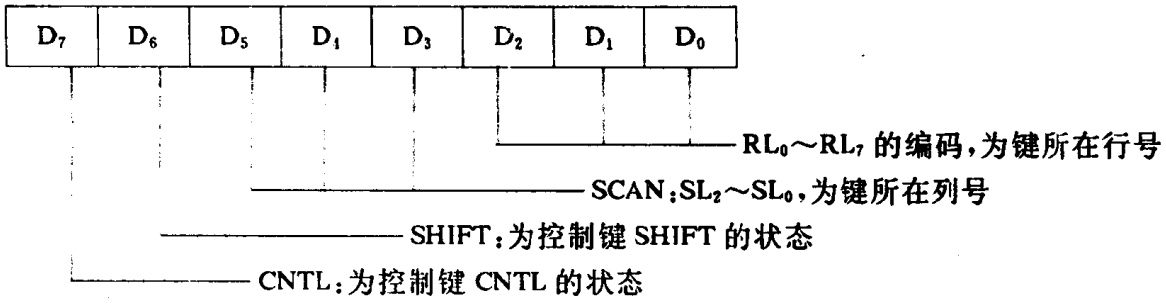
图 3.22　8279 键盘和显示接口电路

8279 的状态字格式和键值格式如下所示。

8279 状态字格式：



8279 键输入数据存放格式：



程序清单如下：

```
;     exam33.asm
;
stack       segment para stack ' stack'
            db 256 dup (0)
stack       ends
data        segment para public ' data'
scantable   db 0,0,' 1234567890-=' ,8,0          ;假设键值表
            db ' qwertyuiop[]' ,0dh,0
            db ' asdfghjdl;' ,0,0,0,0
            db ' zxcvbnm,./' ,0,0,0
            db '  ' ,0,0,0,0,0,0,0,0,0,0,0,0,0,0
            db ' 789-456+1230.'
data        ends
code        segment para public ' code'
start       proc far
;标准的程序框架
            assume cs:code
            push ds
            mov ax,0
            push ax
            mov ax,data
            mov ds,ax
```

```
        assume ds:data
;第一部分:设置自己的键盘中断服务程序
        cli                                ;禁止所有的中断
        mov ax,0
        mov es,ax                          ;指向附加段(同数段地址)
;中断服务子程序地址表
        mov di,28h                         ;类型码 0AH 的偏移地址为 28H
        mov ax,offset kbint                ;键盘中断程序 KBINT 偏移->AX
        cld                                ;地址增加方向存储
        stosw
        mov ax,cs                          ;KBINT 的段地址->AX
        stosw                              ;装入中断向量表中
        mov al,0                           ;写入工作方式命令字
        mov dx,267h                        ;8 字符显示,左入口
        out dx,al                          ;编码扫描键,2 键联
        mov al,2ch                         ;写入时钟分频数 12
        out dx,al
        mov al,0d3h                        ;发清除命令
        out dx,al
        in al,21h                          ;取中断屏蔽寄存器的内容
        and al,0fbh
        out 21h,al                         ;允许 IRQ2 中断
again:  sti
        jmp again
        ret
start   endp
kbint   proc far
        push ax
        push bx
        push dx
        mov dx,267h
        mov al,40h                         ;写入读键控制字
        out dx,al
        dec dx
        in al,dx                           ;读入键盘 RAM
        and al,3fh                         ;屏蔽高 2 位
        mov ah,al
        mov al,20h                         ;中断结束命令
        out 20h,al
        mov al,ah
        lea bx,scantable                   ;取表指针
        xlat                               ;扫描码转换为 ASCII
        cmp al,0                           ;是有效键码?
```

```
        jz exito                        ;否! 忽略它
        call disp
exito:  pop dx
        pop bx
        pop ax
        iret
kbint   endp
disp    proc near                       ;写入显示 RAM
        push dx
        push ax
        mov dx,267h
        mov al,90h                      ;首地址为 0,每写入 1
        out dx,al                       ;位 LED 后,地址自动加 1
        pop ax
        dec dx
        out dx,al                       ;写入 0 号地址段码值
        pop dx
        ret
disp    endp
code    ends
        end start
```

3.4.3 键盘/显示接口实验

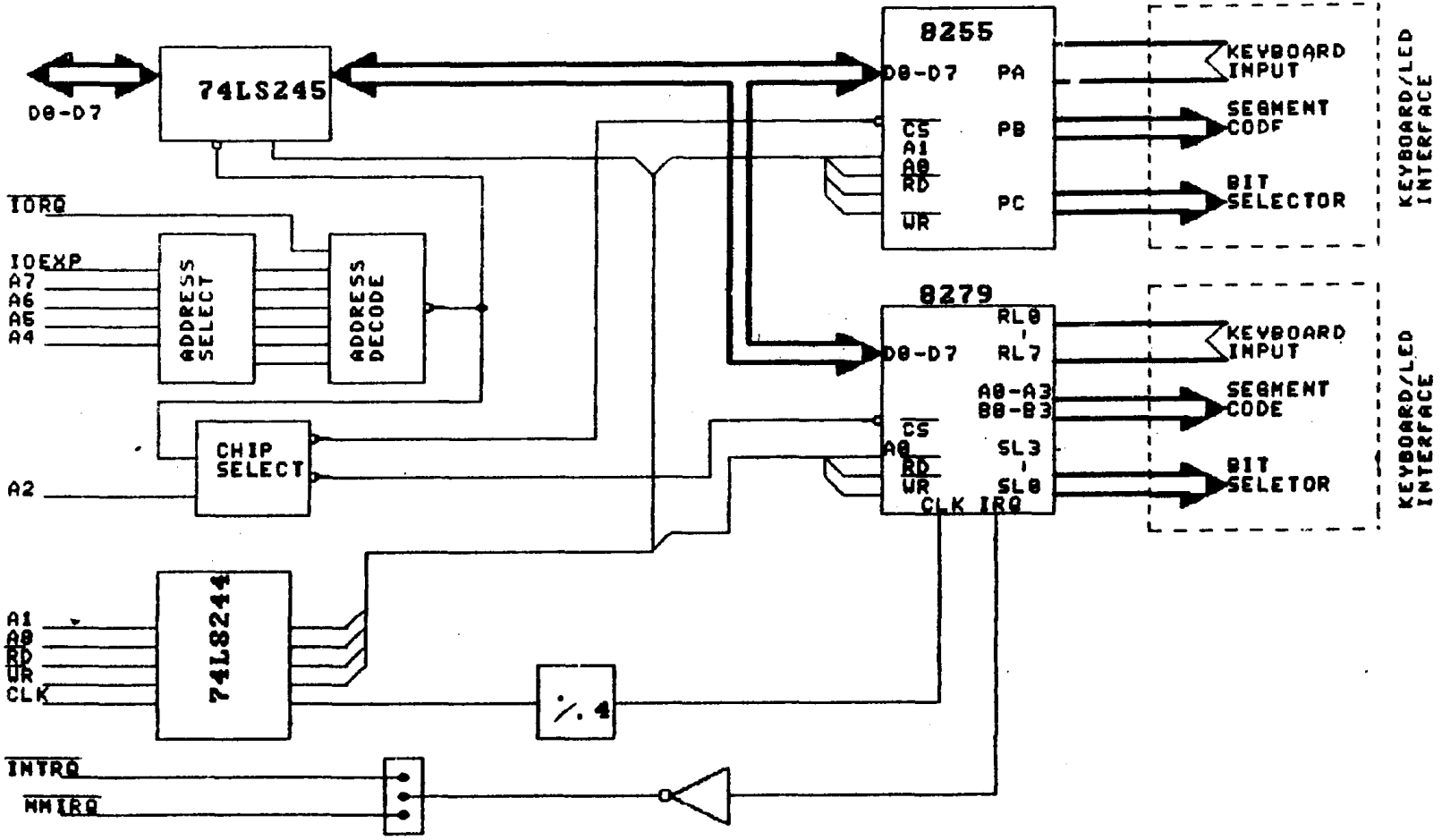
实验 3.3 可编程键盘/显示接口 8279 原理及应用

一、实验原理

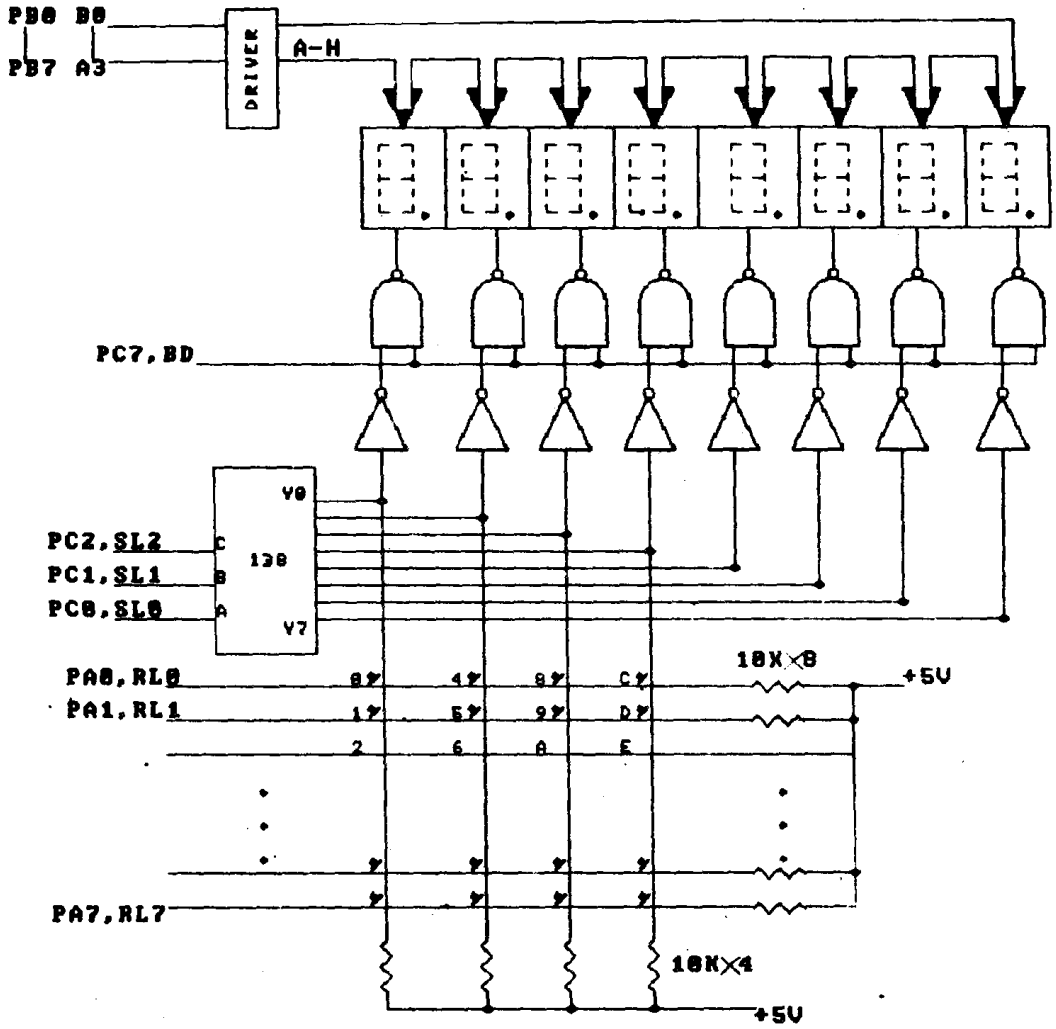
本实验的原理图如图 3.23(a)所示,实验板上有 2 个 26 芯双排插座,均可连接图 3.23(b)所示的 1×8 键盘及 8 位 LED 显示板。并于 Intel 8255 并行接口部分将在 5.3.3 节介绍。

如图 3.23 所示,8279 通过一个 26 芯插座与 4×8 矩阵键盘及 6 位 LED 显示板相联,$B_{0\sim3}$、$A_{0\sim3}$为显示段码,$RL_{0\sim7}$为 8 条键盘扫描的回复线,作为键盘矩阵的行,$SL_{0\sim3}$为 4 位扫描计数器(二进制)的输出,经外部译码器译码后,为键盘和显示器提供扫描线,本实验为 8 位 LED 显示,32 个键只需 4 条扫描列线,因此,只用 $SL_{0\sim2}$,并用 3-8 译码器译码得到 8 条扫描线。显示控制还用了$\overline{BD}$控制输出,用于显示消隐。

8279 与 CPU 的接口如图 3.23(a)所示,$D_0\sim D_7$ 为 8 位数据线,可直接连到数据总线,$\overline{CS}$(低有效)、A_0、$\overline{RD}$、$\overline{WR}$共同完成对 8279 读写操作的控制。CLK 为外部时钟输入端,实验板上的 CLK 来自于主板晶振信号,并经 4 分频后得到(晶振 f_{osc}=6MHz,则 CLK 输入 1.5MHz),在 8279 内部有一个可编程分频器,对 CLK 输入分频后要求为 100kHz。IRQ(高有效)为键盘中断输出信号,8279 一旦检测到回复线 $RL_{0\sim7}$之一为低(没有去抖动延时),便置 IRQ 为高,向 CPU 申请中断。



(a)8255/8279 接口电路原理框图



(b)八位 LED 动态显示及 4×8 矩阵键盘原理图

图 3.23　8255/8279 接口实验原理框图

本实验的键值表如图 3.24 所示。

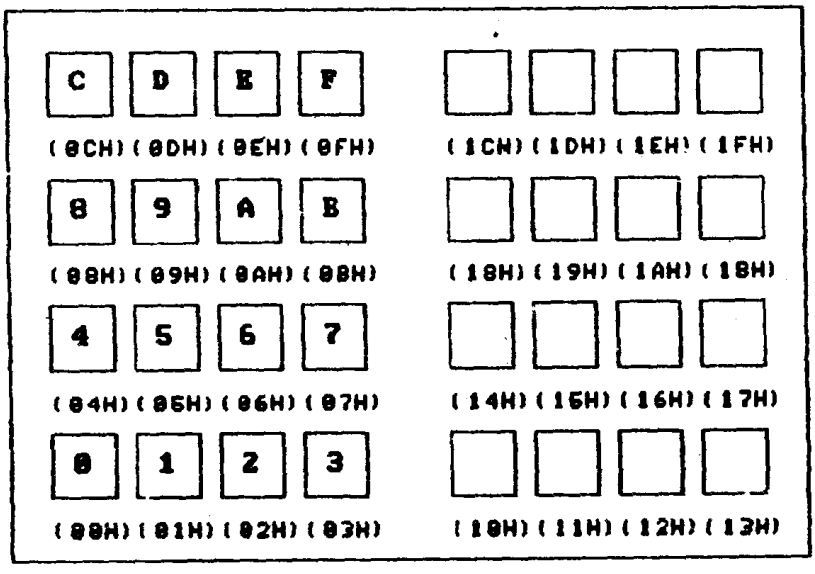
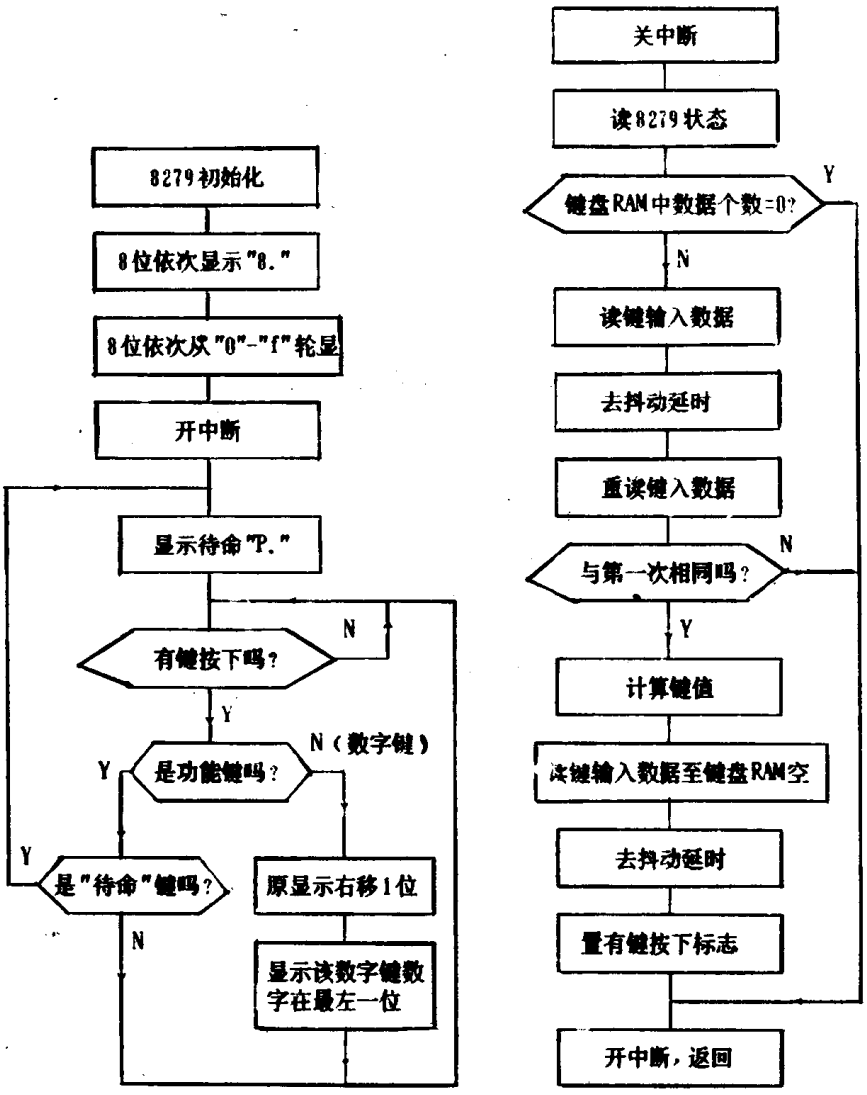


图 3.24　键值表

二、实验要求

1. 根据图 3.23 所示原理框图设计线路图，确定 8279 的数据口和控制口的端口地址。

2. 参考图 3.25 所示的程序流程图设计 8279 键盘与显示实验程序和中断服务程序。



(a)8279 键盘显示程序框图　　(b)中断服务程序

图 3.25　8279 键盘显示程序流程图

程序中的中断服务程序用于读取键值，在中断服务程序中还需加入去抖动的软件延时，若确认是有效键按下时，将读到的键码经过运算得到 00H～1FH 的键值并置有键按下标志，再开中断返回。需要指出的是，在读到有效键后，应继续读键输入数据(这些数据均由按键时抖动引起)直到键盘 RAM 为空(也可用清除键盘 RAM)，否则在中断返回后又将进入中断(因为只要键盘 RAM 中有数据，则中断请求就有效)。

3. 运行程序，并分析所观察到的现象。

3.5 A/D 接口扩展卡的设计

3.5.1 A/D 芯片 ADC0809 的引脚与功能

ADC0809 采用 28 脚双列直插式封装，如图 3.26。各引脚功能如下：

IN_0～IN_7：8 路模拟输入端。输入信号幅度 0～5V，不用的输入端应接地。

ADDA、ADDB、ADDC：8 路模拟开关的选通地址。

ALE：地址锁存允许端。上升沿将 ADDA、ADDB、ADDC 三位地址信号锁存，译码选通对应的模拟通道。

REF(+)、REF(−)：基准电压输入端。要求 $1/2[U_{REF(+)}+U_{REF(-)}]=U_{CC/2}$，其偏差值≤0.1V，因此，在单电源+5V 时，REF(+)接+5V，REF(−)接地。

START：转换启动脉冲输入端。在模拟通道选通地址锁存后，向 START 端加一个正脉冲，进行启动转换，脉冲的上升沿使所有的内部寄存器清零，下降沿使 A/D 转换开始。

CLOCK：片内要求的时钟输入端。要求时钟≤650kHz。

EOC：转换结束信号输出端。在 START 信号之后，A/D 开始转换，EOC 为低电平，表示转换在进行中，在转换结束，数据已锁存在输出三态锁存器后，EOC 变为高电平，表示转换已结束，EOC 可作为被查询的状态信号或中断请求信号。

ENABLE：输出允许端。此端加一高电平，可打开 ADC0809 的输出三态锁存器，使数据送到总线上。

3.5.2 A/D 接口方法

ADC0809 和微机接口是通过联络信号(启动、结束等信号线)。通道地址输入和数据输出线与微机连接。其硬件接口方法如下：

1. ADC0809 是由脉冲启动转换的，当给 START 端加上一个正脉冲，则由该脉冲的下降沿启动一次转换。

2. 启动转换时，EOC 端变为低电平，转换结束时，EOC 端升为高电平。

3. 数据输出寄存器有三态输出功能，当 OE(ENABLE)端为高电平，三态门接通。

4. 8 路模拟输入(IN_0～IN_7)由三位地址码输入控制线(ADDA、ADDB、ADDC)进行选择，ADC0809 在 ALE 正的上升沿时刻由芯片内部锁存这三位地址码。

5. 时钟脉冲需由外部提供。

软件编程可采用如下方法：

1. 选通模拟量输入通道；

2. 发出启动脉冲；

3. 用查询、中断等办法等待转换结束；

4. 转换结束后读取转换结果。

3.5.3 A/D 接口实列

例 3.4 设有 8 路外部模拟输入信号，现对它们进行采样，并将结果依次存放于内存 BUFFER 单元，采收 100 组数据后停止。其电路连接如图 3.27 所示。A/D 转换启动信号 START 和通道选择地址的锁存信号 ALE 相连，以便同时锁存通道地址并开始 A/D 采样转换。其输入控制信号是$\overline{CS}$和$\overline{IOW}$。当 A/D 转换结束后就会自动产生 EOC 信号，此信号接 IRQ_2，从而引起一次硬件中断，在中断处理程序中用 IN 指令即可读取转换结果。程序清单如下：

```
;     exam34.asm

data      segment
buffer    db 80 dup(?)
data      ends
stack     segment para stack ' stack'
          db 256 dup(0)
stack     ends
code      segment
          assume cs:code,ds:data,ss:stack,es:data
main      proc far
start:    push ds
          xor ax,ax
          push ax
          mov ax,data
          mov ds,ax
          mov ax,stack              ;初始化
          mov ss,ax
          mov al,0ah
          mov ah,35h                ;取 IRQ2 对应的中断向量到 ES:BX
          int 21h
          push es
          push bx                   ;保存原中断向量
          push ds
          mov ax,seg sampl          ;取中断处理程序的段地址送 DS
          mov ds,ax
          mov dx,offset sampl       ;偏移量送 DX
          mov al,0ah
          mov ah,25h                ;设置 IRQ2 中断向量指向中
          int 21h                   ;断处理程序
          pop ds                    ;恢复 DS
          in al,21h                 ;取中断屏蔽寄存器的内容送
          mov bp,ax                 ;BP 保存
          and al,11111011b          ;允许 IRQ2 中断
```

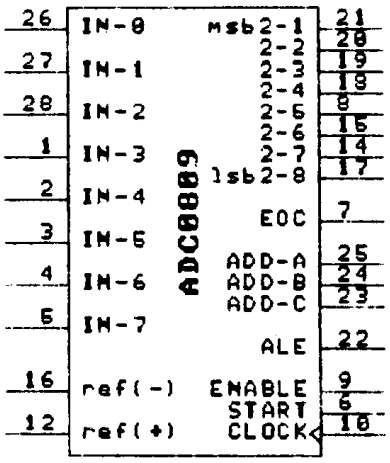


图 3.26 ADC0809 芯片的引脚排例

```
        out 21h,al
        mov ax,data
        mov es,ax
        cld
        lea si,buffer
        mov di,si
        mov cx,10
againo: push cx
        mov al,0                ;转换地址选中 IN
        mov cx,8
again:  sti
        mov dx,263h
        out dx,al               ;送 SC 转换起动信号
        inc al
        hlt                     ;暂停机,等中断,即 EOC 变高
        cli                     ;关中断
        loop again              ;中断返回后,开始采样下一通道
        pop cx
        loop againo
        pop dx
        pop ds
        mov al,0ah              ;恢复 IRQ2 中断向量
        mov ah,25h
        int 21h
        mov ax,bp               ;恢复原中断屏蔽寄存器的内容
        out 21h,al
        ret
main    endp
sampl   proc near               ;中断处理子程序
        push ax                 ;保存
        push cx
        mov dx,263h             ;读取 A/D 转换后的数据
        in al,dx
        stosb                   ;数据送到 BUFFER
        mov al,20h              ;发中断结束命令
        out 20h,al
        pop cx                  ;恢复
        pop ax
        iret
sampl   endp
code    ends
        end     start
```

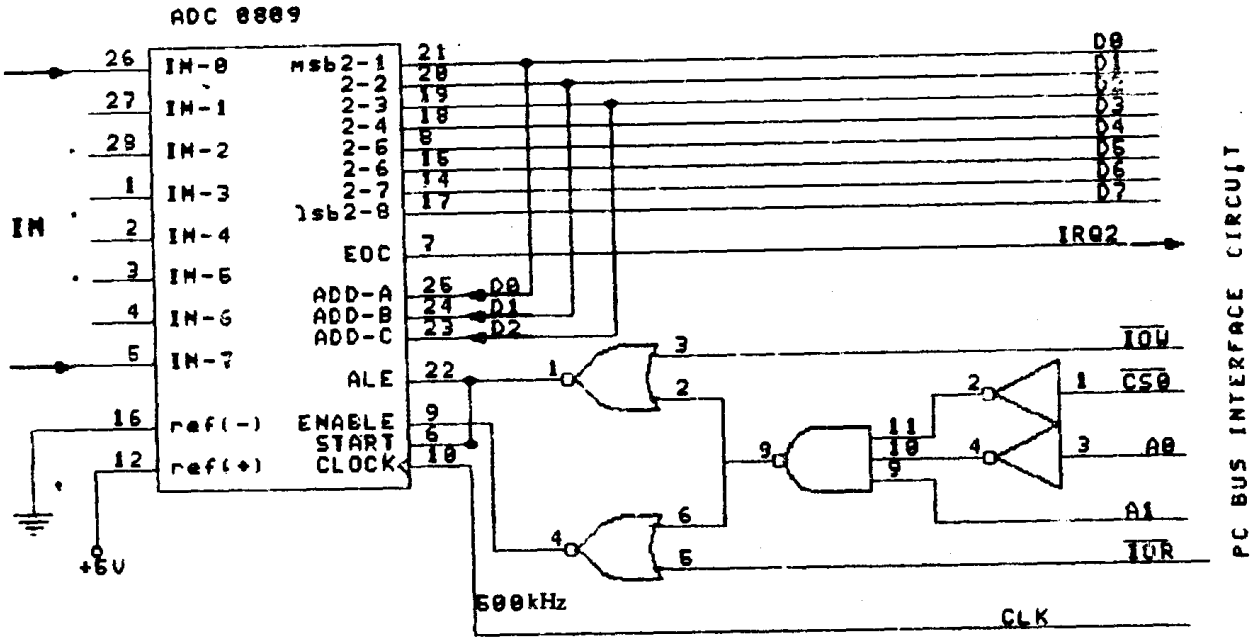


图 3.27　A/D 转换电路框图

3.6　D/A 接口扩展卡的设计

3.6.1　D/A 芯片 DAC0832 的引脚与功能

DAC0832 的管脚排列如图 3.28，其管脚功能如下。

$DI_{0\sim7}$：数据输入端。DI_7 为 MSB，DI_0 为 LSB。

$\overline{CS}$：片选。

$\overline{WR_1}$：对第 1 级缓存的写信号。

ILE：输入锁存允许端。用于对第 1 缓存器进行选通控制。

$\overline{WR_2}$：对第 2 级缓存的写信号。

$\overline{XFER}$：传送控制信号。用于对第 2 级缓存器进行选通控制。

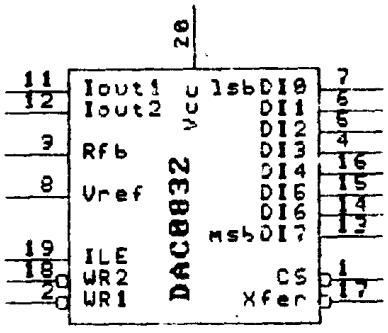


图 3.28　DAC0832 芯片的引脚排列

I_{OUT_1}：DAC 电流输出 1。

I_{OUT_2}：DAC 电流输出 2。

R_{fb}：反馈电阻引出端。为外接运放准备的反馈电阻。

V_{ref}：基准电压输入端。电压为+10V～−10V。

V_{CC}：电源电压，它可以在+5～+15V 内选择，最佳状态是采用+15V 电源。

AGND：模拟地。

DGND：数字地。

3.6.2 D/A 接口方法

一、DAC0832 的硬件接口方法

0832 和 CPU 有三种接口方法，即无缓冲器、单缓冲器和双缓冲器法。

1. 无缓冲器方式

若应用系统中 D/A 转换器不需锁存输入数据或在数据输入端外面已加有输出口锁存数据时，这时 0832 可以工作在内部锁存器直通方式，如同内部无锁存器一样。这时 0832 的 ILE 接+5V，$\overline{CS}$，$\overline{WR_1}$，$\overline{WR_2}$，$\overline{XFER}$四条线接地。如图 3.30 所示。

2. 单缓冲器方式

若应用系统中只有一路 D/A 转换或虽然是多路转换，但并不要求同步输出时，则采用单缓冲器方式接口，让 ILE 接+5V，$\overline{XFER}$及$\overline{WR_2}$都直接与地线相连，$\overline{WR_1}$与 CPU 的$\overline{WR}$相连。即让 DAC 锁存直通，$\overline{CS}$，$\overline{IOW}$选通控制输入锁存器。如图 3.29 所示。

单缓冲方式也可采取另外的接法，即让输入锁存器直通(即 ILE 接+5V，$\overline{CS}$，$\overline{WR_1}$接地)，CPU 选通控制 DAC 锁存器的$\overline{XFER}$，$\overline{WR_2}$。

3. 双缓冲器同步方式

对于多路 D/A 转换接口，要求同步进行 D/A 转换输出时，必须采用双缓冲器同步方式。此时，数字量的输入锁存和 DAC 锁存是分两步完成的，即 CPU 的数据总线分时地向各路 D/A 转换器输入要转换的数字量并锁存在各自的输入锁存器中，然后 CPU 对所有的 D/A 转换器发出控制信号，使各个 D/A 转换器的输入锁存器中的数据打入 DAC 锁存器，实现同步转换输出。

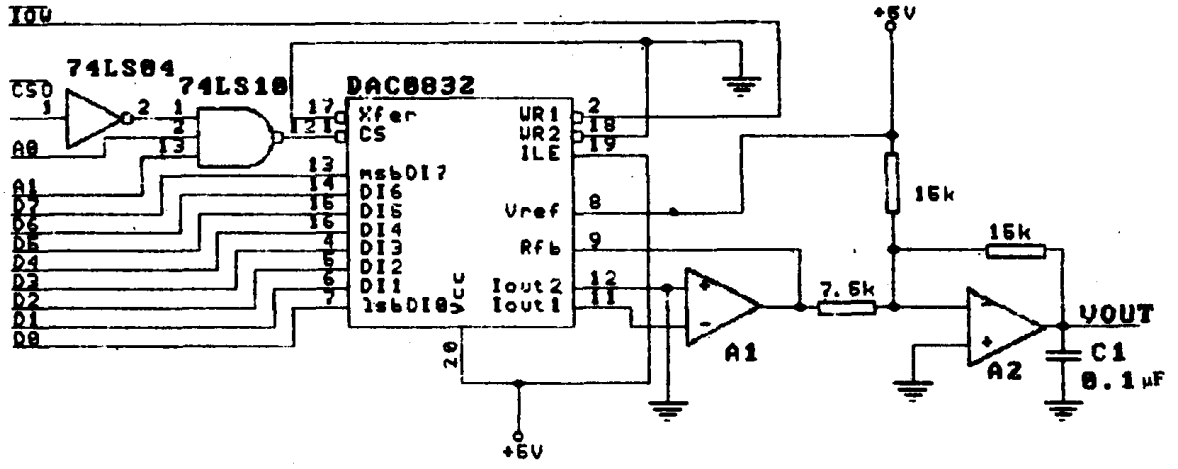


图 3.29 D/A 转换电路框图

二、DAC0832 的软件编程

DAC 能以非常高的速度将加于输入端的二进制数变换成一个输出模拟电压，通过软件编程，由微机向 DAC 输入端提供二进制数据流(一系列随时间变化的二进制数)，则 DAC 产

生一个相应的随时间变化的输出电压(此电压波形可用示波器观测),这就是软件控制 DAC 产生多种波形信号的基本原理。

由微机产生数据流有单路方式和多路方式(二路、三路)两种。当微机只有一个输出数据流时,则 DAC 可产生单一走向的电压波形,如锯齿波、三角波、正弦波(sin(x)),辛格波(sin(x)/x)、钟形波等波形。若微机同时给两路 DAC 输出数据流,它们分别产生对应于示波器 x,y 轴上的输出电压,则能产生复杂的波形和图案。

用软件产生数据流的方法大致有两种:

(1)函数法:根据函数的数学表达式,利用算法程序产生若干基本函数,进而产生相应的电压波形。

(2)数表法:也称数据表法或取样法。根据各种波形的数字表达式或任意曲线的形状,依次取样,列出一个函数表格。

3.6.3 D/A 接口实例

例 3.5 D/A 接口

DAC0832 是一种 8 位的 D/A 芯片。片内有两个寄存器作为输入和输出之间的缓冲。这种芯片可直接挂接到总线上,其连接电路如图 3.29 所示。

图 3.29 中的双极性输出端 V_{OUT}。当 D/A 变换器输入端的数据从 00H～FFH 变化时,V_{OUT} 输出将在 −5V～+5V 之间变化。如果想要单极性 0～+5V 输出,那么只要使 $V_{REF}=-5V$,然后直接从运算放大器 A_1 的输出端输出即可。在图中的输出端接一个 680～6800pF 的电容是为了平滑 D/A 变换器的输出,同时也可以提高抗脉冲干扰的能力。

由于 D/A 芯片是挂接在 I/O 扩展总线上的,因此在编制 D/A 驱动程序时,只要把 D/A 芯片看成是一个输出端口就行了。向该端口送一个 8 位的数据,在 D/A 输出端就可以得到一个相应的输出电压。设 D/A 的端口地址为 263H,则用 8088 汇编语言书写的,能产生锯齿波、三角波和正弦波的程序如下:

```
;       exam35.asm

data    segment
sawtooth db ' outputing sawtooth wave,end with q! ' ,0dh,0ah,' $'
triangle db ' outputing triangle wave,end with q! ' ,0dh,0ah,' $'
sine    db ' outputing sine wave,end with q! ' ,0dh,0ah,' $'
sinebuf db 0,1,2,5,9,13,17,20,23
        db 25,26,25,23,20,17,13,9,5,2,1
data    ends
stack   segment para stack ' stack'
        db 256 dup(?)
stack   ends
code    segment
assume  cs:code,ds:data,ss:stack,es:data
main    proc far
start:  push ds
        xor ax,ax
        push ax
```

```
        mov ax,data
        mov ds,ax
        mov es,ax
        mov ax,stack
        mov ss,ax
        mov dx,offset sawtooth
        call disp                    ;显示提示信息
wave1:  mov cx,0ffffh                ;循环次数
        mov dx,263h                  ;端口地址
again0: mov al,00h                   ;从 00H 开始
again1: out dx,al                    ;启动转换
        add al,1                     ;修改输出数据
        cmp al,20h                   ;数据达到峰值? 不是则
        jnz again1                   ;转 AGAIN1
        loop again0
        mov ah,01h
        int 21h                      ;接受键入字符,判断是' Q'
        cmp al,' q'                  ;否?
        jz exit1
        cmp al,' Q'
        jnz wave1                    ;不是则继续显示锯齿波形
exit1:  mov dx,offset triangle
        call disp                    ;显示提示信息
wave2:  mov cx,0ffffh                ;循环次数
        mov dx,263h                  ;0832 输入寄存器端口地址
        mov al,00h                   ;从 00H 开始
again21: out dx,al                   ;数据送输入寄存器,启动转换
        add al,1                     ;修改输出数据
        cmp al,10h                   ;到三角波波峰否?
        jnz again21
again22: out dx,al                   ;送输入寄存器,启动转换
        sub al,1                     ;修改输出数据
        cmp al,00h                   ;到三角波波谷否?
        jnz again22
        loop again21                 ;循环
        mov ah,01h
        int 21h                      ;接受键入字符,判断是' Q'
        cmp al,' q'                  ;否?
        jz exit2
        cmp al,' Q'
        jnz wave2                    ;不是则继续显示三角波
exit2:  mov dx,offset sine
        call disp                    ;显示提示信息
```

```
wave3:   mov cx,0ffffh                ;循环次数
         mov dx,263h
again31: mov si,offset sinebuf        ;建立数据表首地址
         mov bl,20                    ;共 20 个数据
again32: mov al,[si]                  ;数据送输入寄存器,启动转换
         out dx,al
         inc si                       ;指针 SI 指向数据表下一单元
         dec bl                       ;已显示完一个周期间 20 个点
                                      ;的数据?
         jnz again32
         loop again31                 ;循环
         mov ah,01h
         int 21h                      ;接受键入字符,判断是' Q' 字
                                      ;符否?
         cmp al,' q'
         jz exit3
         cmp al,' Q'
         jnz wave3
exit3:   ret
         main endp
disp     proc near
         push ax
         push dx
         mov dl,0dh
         mov ah,02h
         int 21h
         mov dl,0ah
         mov ah,02h
         int 21h
         mov ah,09h
         int 21h
         pop dx
         pop ax
         ret
disp     endp
code     ends
         end      start
```

例 3.6 A/D 和 D/A 应用实例

设有一系统,要求监控 8 路参数,每一路的转换精度要求为 8 位,当这 8 路某一刻的值达到某种关系时,则要求输出一模拟控制信号到系统。为此,根据上述思想,便设计出接口电路的框图如下(图 3.30),其中设 A/D,D/A 都采用 8 位,应用 ADC0809 和 DAC0832,而并行接口芯片则应用 8255。

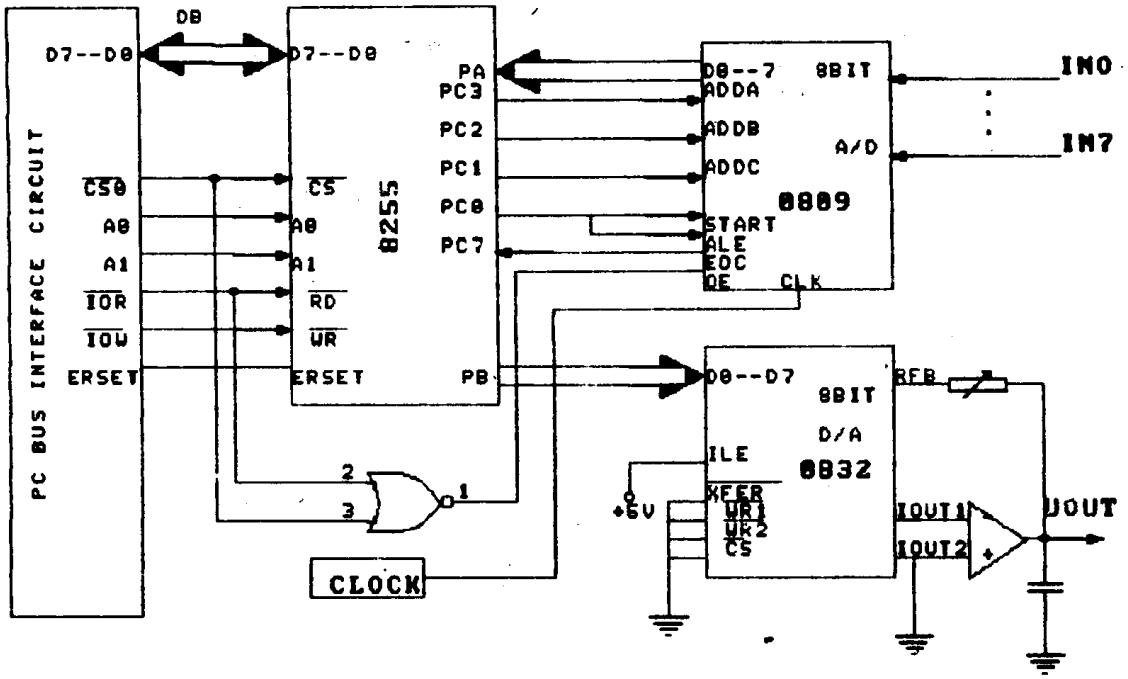


图 3.30　A/D 和 D/A 转换接口电路框图

图中 8255 的端口 A 工作于方式 0，并设定为输入操作。端口 C 为低 4 位设定为输出操作，与 A/D 转换器的启动信号 START 和输入地址 ADDA、ADDB、ADDC 相连，也即由 PC_0 位提供一个启动脉冲，PC_1，PC_2，PC_3 提供模拟输入信号的选择地址，ADC 的 EOC 信号作为“忙”信号提供给 PC_7；以供程序查询是否转换结束可以读数；端口 B 设定为方式 0 的输出方式，直接与 DAC 的输入相连接，输出控制信号。DAC 采用直通型连接方式，即片选信号$\overline{CS}$和写控制信号$\overline{WR_1}$、$\overline{WR_2}$、$\overline{XFER}$等都接地。8255 的端口 A、B、C 及控制寄存器的 I/O 地址分配为 260～263H。现需采集 8 位模拟信号输入，如果前面 7 路(u_0～u_6)之和大于等于第 8 路(u_7)，则输出一三角波信号，要求不低于 500 个周期，否则继续采样。程序清单如下：

```
;       exam36.asm

                ;progam to A/D D/A
                ;port address of 8255 are 260h－263h
                ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
sseg            segment para stack ' stack'
                dw 128 dup(0)
sseg            ends
dseg            segment
waitt           db ' A/D D/A example program' ,0dh,0ah,' $'
strike          db ' STRIKE ANY KEY to begin sample data...' ,' $'
buf1            db 7 dup(0)
buf8            db 0
porta           equ 260h
portb           equ 261h
```

```
portc       equ 262h
portl       equ 263h
sampim      db ' Sampling data....' ,' $'
otwave      db ' OUTPUT TRIANGLE WAVE...' ,' $'
dseg        ends
cseg        segment
main        proc far
            assume cs:cseg,ds:dseg,ss:sseg,es:dseg
            push ds
            xor ax,ax
            push ax
            mov ax,dseg
            mov ds,ax
            mov es,ax
            cld
            call init               ;8255 初始化
            call clear              ;清屏幕
            mov dx,0a30h            ;设置光标位置(10,48)
            call cursor
            lea si,waitt
            call disp               ;显示 WAITT 字符串
            mov dx,1230h            ;设置光标位置(18,48)
            call cursor
            lea si,strike
            call disp               ;显示 STRIKE 字符串
            mov ah,0                ;等待处理
            int 16h
            call clear              ;清屏
            mov dx,0c30h
            call cursor             ;设置光标(12,48)
            lea si,sampim
            call disp               ;显示 SAMPIM 字符串
again:      lea di,buf1             ;建立地址指针
            mov cx,8
            call sampl              ;采集 8 路模拟量并送 BUF1
            lea dx,buf1
            mov cx,7
            call addt               ;求 1—7 路模拟量累加和,并放在 AX 中
            xor bh,bh
            mov bl,buf8
            cmp ax,bx               ;如果前 7 路累加和不大于第 8 路,则继续采集
            jle again
            mov dx,0c30h            ;设置光标位置(12,48)
            call cursor
```

```
        lea si,otwave          ;显示 OTWAVE 字符串
        call disp
        call daout             ;输出三角波信号
        ret
main    endp
clear   proc                   ;清屏幕
        mov ax,0600h
        mov bx,14
        mov cx,0h
        mov dx,184fh
        int 10h
        ret
clear   endp
cursor  proc                   ;设置光标位置
        mov ah,2
        mov bh,0
        int 10h
        ret
cursor  endp
disp    proc                   ;显示 SI 指向的字符串
disp1:  lodsb
        cmp al,'$'
        je rett
        mov ah,09
        mov cx,1
        mov bh,0
        mov bl,28
        int 10h
        mov ah,0eh
        int 10h
        jmp short disp1
rett:   ret
disp    endp
init    proc
        mov dx,portl           ;A 口方式 0,输入,C 口高 4 位输入
        mov al,10011000b       ;B 口方式 0,输出,C 口低 4 位输出
        out dx,al
        ret
init    endp
sampl   proc                   ;数据采集
        mov al,00000001b       ;发 A/D 转换启动信号
nextt:  mov dx,portc
        out dx,al
```

```
        push ax
        and al,11111110b
        out dx,al
waiter: in al,dx                ;等待转换结束
        test al,10000000b
        jz waiter
        mov dx,porta            ;从 A 口读数据
        in al,dx
        stosb                   ;存数据到 BUF1
        pop ax
        add al,02               ;选择下一路模拟输入
        loop nextt
        ret
sampl   endp
addt    proc                    ;求 1—7 路模拟量的累加和
        lea si,buf1
        mov dx,0
ad02:   lodsb
        add dl,al
        adc dh,0
        loop ad02
        mov ax,dx
        ret
addt    endp
daout   proc                    ;输出三角波信号
        mov dx,portb
        mov cx,500
wave:   mov al,0
lopx:   out dx,al
        inc al
        jnz lopx
lpxo:   dec al
        out dx,al
        jnz lpxo
        loop wave
        ret
daout   endp
cseg    ends
        end main
```

3.6.4 A/D 和 D/A 接口实验

一、实验原理

A/D 及 D/A 接口的硬件电路如图 3.31 所示，主要由 D/A 转换芯片 DAC0832(二片)和

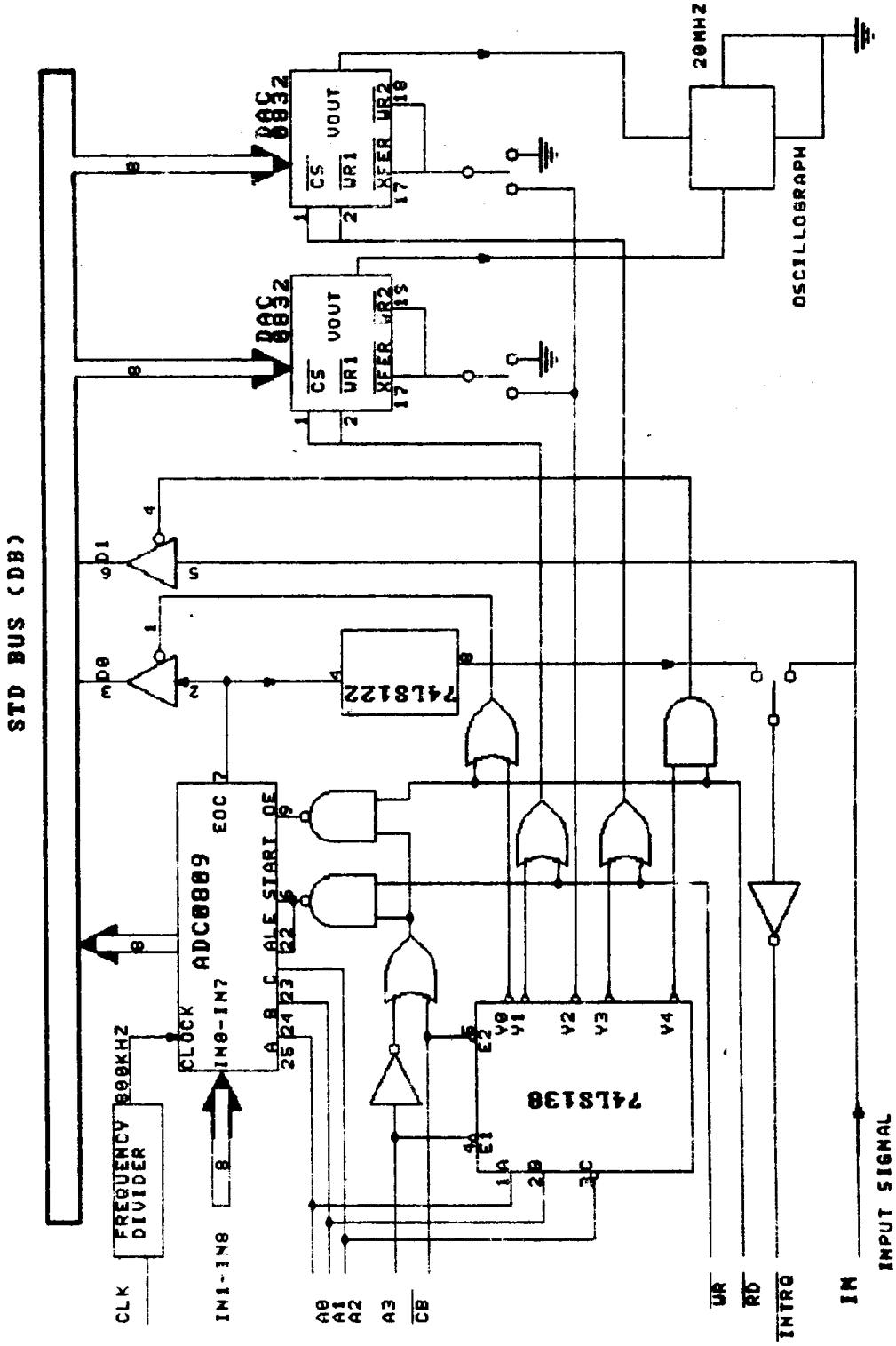


图 3.31 A/D 和 D/A 转换实验原理框图

A/D 转换芯片 ADC0809(一片)构成。

1. A/D 转换电路

A/D 转换电路由 A/D 转换芯片 ADC0809 为主构成,8 路模拟量输入由三位地址线 A_0～A_2 进行选择。ADC0809 的 START 端与 ALE 端相连,以便同时锁存通道地址并开始 A/D 采样转换。其输入控制信号为板选信号$\overline{CB}$、地址信号 A_3 和$\overline{WR}$,故启动 A/D 转换只须用 OUT 指令向 ADC0809 的端口发送任意数据即可,具体发送数据的值是无关紧要的,因为这是一次虚写,目的是产生 A_3、$\overline{CB}$及$\overline{WR}$转换启动控制信号。

当 A/D 转换结束后,自动产生 EOC 信号,此信号反相后接$\overline{INTRQ}$从而引起一次硬件中断,在中断处理程序中,使用 IN 指令即可从 ADC0809 的端口读取 A/D 转换结果,因为输出允许信号 OE 是由读信号$\overline{RD}$、板选信号$\overline{CB}$和地址信号 A_3 控制的。

2. D/A 转换电路

本实验板上提供了两路 8 位的 DAC,因此能作单路输出的多波信号的实验,也能进行双路输出的二维图形实验。DAC0832 可接成单缓冲方式,也可接成双缓冲方式。若用跳线把$\overline{XFER}$(17 脚)和$\overline{WR_2}$(18 脚)接地,则工作在单缓冲方式;若用跳线把$\overline{XFER}$和$\overline{WR_2}$接在 74LS138 译码器的 Y_2 端由译码信号 y_2 来选通 DAC 锁存器,则工作在双缓冲方式。输出采用两级运放构成双极性输出电路,如图 3.32 所示。当数字量输入 00H～FFH 范围变化时,模拟电压输出为－5V～＋5V。

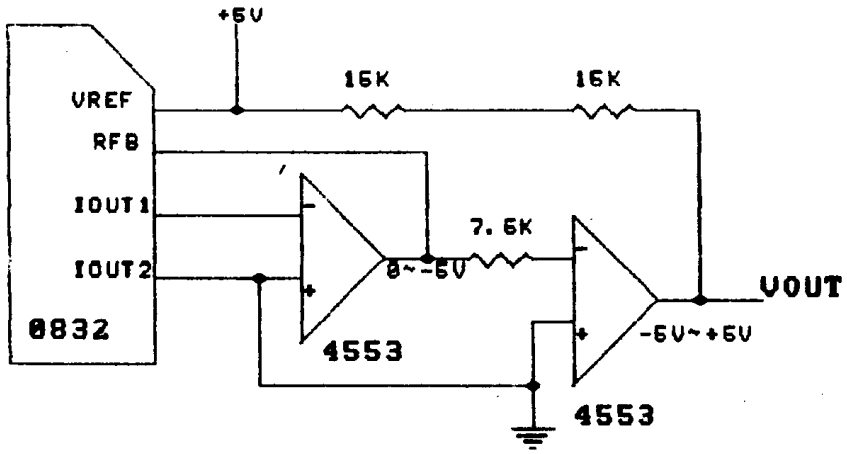


图 3.32　ADC0832 的模拟电压输出电路

二、实验要求

1. 根据图 3.31 和图 3.32 所示的原理框图设计线路图,确定 ADC0809 和 DAC0832 的端口地址。

2. 编写用 DAC0832 产生方波、锯齿波、三角波、正弦波等信号的程序,并在 PC 机上运行程序,用示波器观察输出电压波形。

3. 用 DAC0832 产生一直流信号,送往 ADC0809 某一路输入端。在 DAC 输出端接入一检测用万用表。编写程序完成如下功能。

(1)给 DAC_1 输入一个数字量 N_1,则得到一个输出电压 V_{out_1}(注意,只输出正电压值)。

(2)将相应模拟输入进行 A/D 转换的数字结果处理后，向 DAC_2 输出数字量 N_2，则得到一个输出电压 V_{out_2}。

试比较各数据点的 V_{out_1} 和 V_{out_2} 值，分析测量误差。

第四章　微机应用系统常用接口

4.1　多功能接口卡设计

4.1.1　多功能 I/O 接口

随着微型计算机的应用范围越来越大，微机所带外设已由单一发展到多样化，一台微机所配置的不同特性的 I/O 接口扩展卡也逐渐增多，这不仅使接口成本随之增加，而且有时出现 I/O 扩展槽不够使用。为此，出现了将多种接口控制功能集成在一个 I/O 接口扩展卡上的多功能 I/O 接口扩展卡。

多功能接口扩展卡一般有以下两种组成方式：

(1)用多个不同功能的接口控制器芯片组合而成。它们共用一套地址译码、读/写控制和数据缓冲电路与主机连接。与外设的连接部分，分别采用不同特性的物理接口。

(2)用多个不同接口控制器芯片或单片机实现。它们以软件重构方式实现多功能接口控制特性，与主机连接部分与普通 I/O 接口芯片类似，在用单片机实现时采用主从式 CPU 连接方式。与外设连接部分也分别采用不同特性的物理接口。

尽管市面上有各种各样的多功能接口卡，但要想找到性能合适而又价格便宜的接口卡则很困难。实际上，只要了解 IBM 扩充插槽上各种信号的技术特性和地址译码电路的工作原理，以及有关的接口芯片(如并行接口、A/D 转换等)的性能，就可以自己动手设计一种完全合用的接口卡。

4.1.2　多功能接口卡实例

例 4.1　多功能接口卡

图 4.1 所示是一个具有多通道模拟(电压)信号的采集，多路温度检测，实时控制和并行 I/O 口的扩充等多项功能的接口卡。它主要由 I/O 口地址译码电路、并行接口(8255A)、定时/计数器(8254)、A/D 转换器(ADC0801)、V/F 转换器(LM131)、双 4 路模拟开关(CD4052)和 4 路光电隔离电路等组成。

下面根据其功能分几部分加以说明。

(1)I/O 地址译码电路

该电路简单实用，只由一片 4 输入与非门 74LS20 和一片 3-8 译码器 74LS138 组成。138 的输出 y_0，y_1 和 y_2 作为片选信号分别接 8255A、8254 和 ADC0801 的$\overline{CS}$端，加上 A_0 和 A_1 的口地址译码。上述几个芯片的 I/O 口地址分别为：

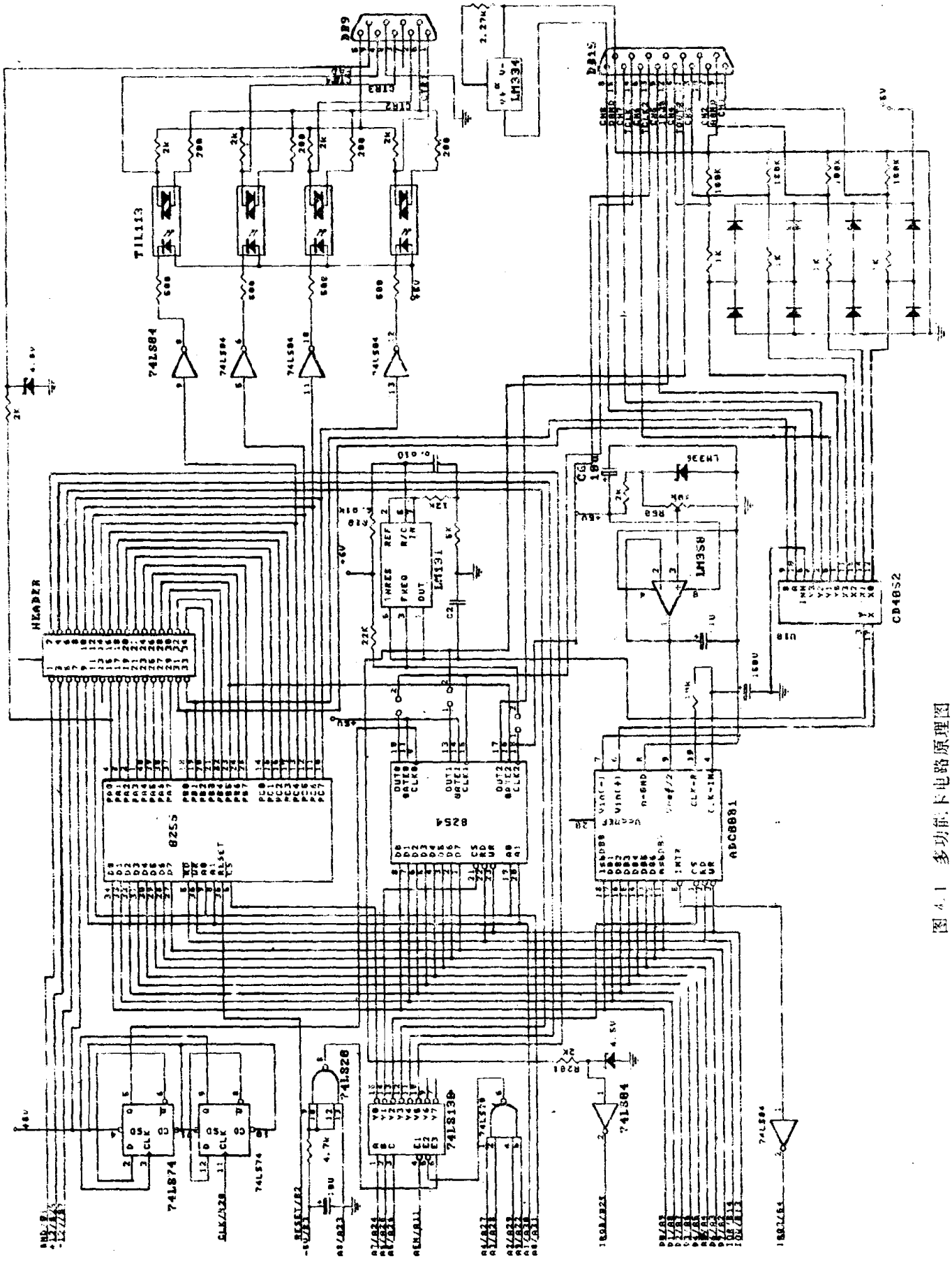


图 4.1 多功能卡电路原理图

8255A		8254	
A 口	21CH	计数器 0	29CH
B 口	21DH	计数器 1	29DH
C 口	21EH	计数器 2	29EH
控制字口	21FH	控制字口	29FH

ADC0801： 25CH

(2)双 4 通道信号选通电路

该电路主要由双 4 通道模拟开关 CD4052 和 8255A 的 PB_0 和 PB_1 组成。PB_0 和 PB_1 接 CD4052 的通道选通控制端 A 和 B，所以信号输入通道的选通是由 PB_0 和 PB_1 来控制。在 CD4052 的 x_0～x_3 输入端接有限幅二级管限制输入信号的幅度，起保护作用。

(3)模拟/数字转换电路

该电路由 ADC0801 和精密可调稳压源组成。精密稳压器件 LM336 和运算放大器 LM358 构成的精密可调稳压源为 ADC0801 的 VREF 端提供非常稳定的参考电压，可调的参考电压用作配合不同幅度范围的输入，以便取得最佳的转换精度。

(4)电压/频率转换电路

V/F 转换器 LM131 及其外围电路实际构成一个 A/D 转换电路。如图 4.1 所示，该电路通过 CD4052 的 y 通道外接一个 LM334 构成一个温度测量电路，LM334 是精密恒流源，外形象一只小三极管，它的电流值与温度变化有很好的线性关系，所以可作为性能优良的温度传感器。有必要说明的是，LM334 是外接器件，为帮助说明才把它标在图 4.1 中。

(5)定时/计数电路

一片双 D 触发器 74LS74 和一片 8254 组成了定时/计数电路，前者接成 4 分频电路的形式，将来自系统总线的 4.77MHz 的 CLK 信号变成 1.19MHz 的信号，以便取得较长的定时时间，此信号接至 8254 的 CLK0 端，作为计数器 0 的计数脉冲。计数器 0 的输出可通过一个跳线开关与计数器 1 的 CLK1 相连，因此两个计数器可串联使用。计数器 1 的输出经过一个跳线开关和一个反相器与系统总线上的中断请求信号 IRQ_3 相连。IRQ3 在中断系统中原分配给串行口 2 用，只要用户没有使用串行口 2，8254 的计数器 1 就可以利用 IRQ_3 申请中断。计数器 2 的 CLK2 可通过一个跳线开关与 V/F 转换器的输出脉冲相连，而 G_2 则由 8255A 的 PB_5 来控制，使计数器 2 的使用更加灵活。此外，8254 的 CLK1，CLK2 和 OUT2，还有来自系统总线的 IRQ_3 信号都可以接至联接器 DB_{15}，供用户检测或外接信号用。

(6)并行口和光电隔离电路

8255A 的 PC_4～PC_7 用于输出控制信号，它们经反相器 74LS04 驱动，再接光电耦合器 TIL113 形成四路光电耦合隔离电路，该电路既能保护计算机，又能增加输出信号的驱动能力。8255A 的 3 个端口的全部引脚、地址译码输出 y_3～y_5，系统总线中的 A_0、A_1、+5V、+12V 和 −12V 都接到一个 34 脚的插座上，供用户通过电缆引至机箱外作进一步的扩充用。

4.2 高速大容量数据采集接口

4.2.1 数据采集接口

一个基本的数据采集接口包括多路模拟开关、测量放大器、抽样保持电路和 A/D 转换

器。如图 4.2。

多路模拟开关用于分时选通不同的传感器送出的信号以复用一个 A/D 通道。目前常用的有 AD7501、AD7502、AD7506、AD7507、CD4051、CD4052、CD4067、CD4097 等。

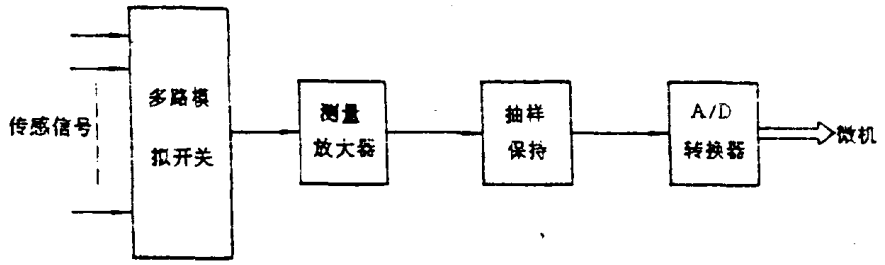


图 4.2 基本的数据采集接口

测量放大器是一个精密的差分放大器，具有很高的抑制共模干扰的能力。目前常用的芯片有 AD521、AD522、INA102 等。抽样保持电路可用 CMOS 模拟开关 CD4501 和运放构成，也可采用专门的抽样保持器芯片，如 AD582 和 LF398 等。

4.2.2 高速大容量数据采集

一、高速大容量数据采集

在实时测量与控制中，当被测对象的信号频率很高时，就需要采用高速数据采集系统。为了实现高速采样，有两个因素应考虑：A/D 转换器的转换时间和转换后的数据传送到主存储器（或外存储器）的时间。若转换时间为 t_1，数据传送时间为 t_2，则采集速率，即每次采样时间为 $t=1/(t_1+t_2)$。可见，要提高采集速度，一方面要选用高速 A/D 转换器；另一方面是要缩短数据传送过程中的时间。

大容量是相对计算机内存大小而言的，数据量超过内存容量，即称为大容量。数据总量超过内存容量，必然要转储于外存，如磁盘。通常 PC 机在内存数据写磁盘过程中，CPU 不能再同时执行数据采集程序，于是数据采集工作便不能继续下去，从而导致一部分数据丢失。这便是高速大容量数据采集的症结所在。要完成高速大容量的数据采集任务，就必须保证在数据写磁盘的同时仍能连续不断地采集数据，不发生数据丢失的问题。

高速大容量数据采集的难办之处，在于如何做到内存数据写磁盘的同时仍能连续不断地采集数据，不发生数据丢失的问题。PC 机所具有的接口资源和功能，为解决这一问题提供了良好的环境。PC 机中内存和磁盘之间的数据传送是用 DMA 方式进行的。PC 机使用 8237-5DMA 控制器，共有 4 个通道，4 个通道的使用分配为：通道 0 用于动态 RAM 刷新，通道 1 留给用户使用，通道 2 和通道 3 分别用于内存与软、硬盘之间的数据传送。PC BIOS 要对 8237-5DMA 控制器进行初始化，使通道 0 的优先级最高，通道 1、2、3 的优先级依次降低，通道 3 的优先级最低。通常使用 DOS 系统功能调用来执行磁盘的读写，但在读写磁盘期间，CPU 不能再运行其他程序。由此可见，要想在读写磁盘期间仍能继续采集数据，就不能指望 CPU，必须另辟蹊径。只要能提供符合 A/D 转换器要求的触发信号，便可使 A/D 转换继续进行，这点可通过增加少量硬件的办法来做到。至于转换好的数据，则可使用 DMA 通道 1 将其输入内存。因为通道 1 的优先级比通道 2 和通道 3 都高，所以把 A/D 转换器的数据输入内存与把内存数据写入磁盘相比，前者优先得到保证。此外 DMA 方式传送数据速度快，正

合高速数据采集对数据传送速度要快的要求，可谓一举两得。

二、DMA 接口芯片 8237 的引脚与功能

8237 芯片是 40 脚封装的双列直插式组件，如图 4.3。各引脚的功能如下：

CLK：时钟输入。8237 为 3MHz，8237-5 为 5MHz。

Reset：复位。它清除所有的寄存器，芯片处于从属状态。

$\overline{CS}$：片选。在从属状态下，该信号开放 I/O 读写；在主控状态下，它被自动禁止，以防止在执行 DMA 操作时，芯片自己选中自己。

AEN：地址允许。这个输出信号用于禁止非 DMA 器件 DMA 周期内产生响应。

DRQ_0～DRQ_3：0～3 通道的 DMA 响应线，若不是处于循环优先级方式，则 DRQ_0 的优先级最高，DRQ_3 的优先级最低。

$DACK_0$～$DACK_3$：0～3 通道的 DMA 响应线，它通知外设，已被允许执行一个 DMA 周期。

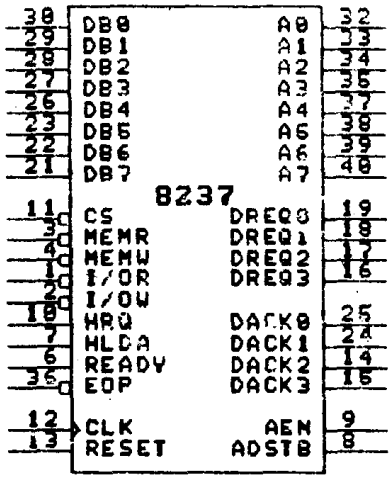


图 4.3　8237 芯片的引脚排列

HRQ：控制总线请求线。这个输出信号用于请求系统总线的控制。

HLDA：放弃总线响应线。这个输入信号表明，8237 已得到系统总线的控制权。

A_0～A_7：低 8 位地址线。A_0～A_3 为双向，在从属状态下，它为输入，作为内部寄存器的地址；在主控状态下，它为输出，作为地址总线的低 4 位。A_4～A_7 为三态单向输出。

DB_0～DB_7：数据线。在从属状态下，它的双向三态数据线，用于 CPU 对 8237 的编程和读取状态；在主控状态下，它分时复用作为高 8 位地址线。

ADSTB：地址选通。该输出信号用于从数据线上分离出高 8 位地址。

$\overline{IOR}$：I/O 读。双向三态线。在从属状态下，它是输入，用于将所寻址的寄存器内容读出到总线上；在主控状态下，它是输出，用于在 DMA 写周期中，从外设读取数据。

$\overline{IOW}$：I/O 写。双向三态线。在从属状态下，它是输入，用于将数据总线的内容写到所寻址的寄存器（用于初始化）；在主控状态下，它是输出，用于在 DMA 读周期中将数据写入外设。

$\overline{MEMR}$：存储器读。三态输出。在 DMA 读周期中，用于从被寻址的存储单元中读取数据。

$\overline{MEMW}$：存储器写。三态输出。在 DMA 写周期中，用于将数据写到被寻址的存储单元。

READY：准备好。如果所选的存储器需要较长的存取周期，则这个输入信号可用于插入等待，从而延长 8237 的存储器读写周期。

$\overline{EOP}$：过程结束。双向信号。当外部要中止正在执行的 DMA 服务时，可输入一个低电平信号；当任何一个通道到达终点计数（T/C）时，EOP 输出一个低电平信号。因此，不论是外部输入的或内部输出的$\overline{EOP}$，都会引起 8237 中止服务，清除请求。

4.2.3　高速大容量数据采集接口实例

高速大容量数据采集系统原理如图 4.4 所示。它采用 12 位 A/D 转换器 AD574A，与软

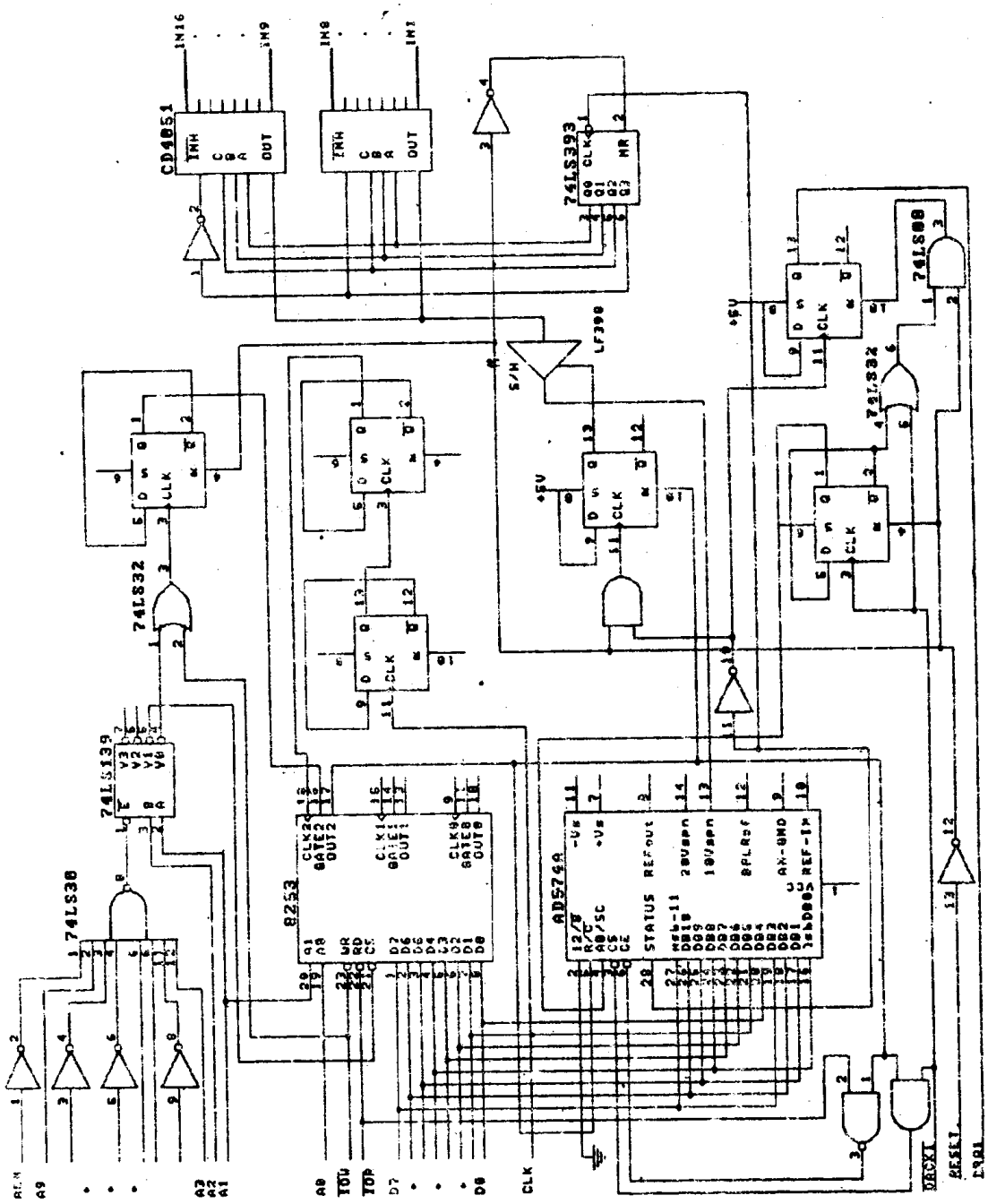


图 4.4　高速大容量数据采集系统原理图

件部分相配合，能连续不断地采集数据，又能同时把已输入内存的数据分批写入磁盘。数据容量完全不受内存大小控制，并具有很高的数据传送速度。系统可分成下列几部分，分别叙述如下：

(1)定时电路。采用 8253-5 可编程计时器，它共有 3 个通道，系统只用其中的通道 2。通道 2 的工作模式设置成方式 2，用以定时产生负脉冲。负脉冲是用来启动 AD574A 进行模数转换的，其发生周期可编程设定。PC 机的 CLK 信号经四分频后为 1.19MHz，供 8253—5 作为时钟输入。当 GATE2=1 时，8253-5 的通道 2 开始计时，采集系统开始工作。

(2)转换电路。由 AD574A 和采样保持器 LF398 组成。AD574A 是 12 位逐次逼近型模数转换器，转换时间为 25μs，带有三态缓冲器，输出时可以一次读出全部 12 位数据，也可按两次 8 位方式读，能很方便地与 8 位或 16 位微机相连接。8253-5 输出的负脉冲一方面使 AD574A 开始进行转换，另一方面又使 D 触发器复位，从而使 LF398 处在保持状态。当转换结束时，$\overline{STS}$的上升沿使 D 触发器复位，LF398 处在采样状态。

(3)16 通道切换电路。由 2 片八选一模拟多路开关 CD4051 及四位计数器 74LS393 组成。每次转换结束时，STS 的下降沿使计数器加 1，自动转换到下一模拟通道，从而使 16 路模拟信号依次循环接通。

(4)DMA 请求电路。由两个 D 触发器组成。每次转换结束时，$\overline{STS}$的上升沿使 $DRQ_1=1$，发出 DMA 通道 1 的请求。DRQ_1 被认可后要进行二次 DMA 传送，先读取 AD574A 的高 8 位数据，再读取低 4 位数据。当$\overline{DACK_1}$第二次变低，即开始读取低 4 位数据时，$DRQ_1=0$，DMA 通道 1 的请求被撤销。

(5)地址译码电路。由八与非门 74LS30 和二-四译码器 74LS139 组成。地址 3EAH 和 3EBH 用于对 8253-5 初始化编程，地址 3E8H 或 3E9H 用于启动系统工作和停止系统工作。

软件部分见数据采集程序清单。该程序可在采集数据的同时将内存数据一批一批地写入磁盘。此程序是针对写入软盘编制的，写入时 A、B 两盘轮流工作，即 A 盘写满写 B 盘，B 盘写满写 A 盘，依次轮换进行。将数据写入软盘的优点在于所采集的数据总量不受任何限制，只要有足够多的完好盘片即可。程序清单如下：

```
;     exam41.asm

stack     segment para stack ' stack'
          db 256 dup(?)
stack     ends
code      segment
          assume cs:code,ss:stack
start     proc far
begin:    push ds
          xor ax,ax
          push ax
          mov dx,26bh                    ;设定 8253 计数器 2 的工作模式，方式 2
          mov al,0b4h
          out dx,al
          dec dx
```

```
        mov al,50h              ;设定计数值低 8 位
        out dx,al
        mov al,00h              ;设定计数值高 8 位
        out dx,al
        mov al,55h              ;设定 DMA 通道 1 的工作模式,单字节写,自
        out 0bh,al              ;动预置
        mov al,00h              ;清高低触发器
        out 0ch,al
        out 02h,al              ;通道 1 的起始地址低八位
        out 02h,al              ;通道 1 的起始地址高八位;
        mov al,0ffh             ;通道 1 传送字节数低 8 位
        out 03h,al
        mov al,0efh             ;传送字节数高 8 位
        out 03h,al              ;传送字节数为 60K
        mov al,02h
        out 83h,al              ;通道 1 的页面值(A 缓冲区,段地址为 2000H)
        out 0eh,al              ;清屏蔽寄存器
        dec dx
        out dx,al               ;起动采集系统工作
        mov ax,0ee01h
        mov bx,ax
loop0:  mov dx,0000h
loop1:  in al,08h               ;读 DMA 状态寄存器
        and al,02h
        jz loop1                ;未达 60K 则数据仍送 A 缓冲区,否则送 B 缓冲
                                ;区,并将 A 缓冲区的数据存盘
        ror bh,01h
        ror bh,01h
        mov al,bh
        and al,03h
        out 83h,al              ;更换页面值(B 缓冲区,段地址为 3000H)
        and ax,3000h            ;形成数据段地址
        mov ds,ax               ;(2000H 与 3000H 轮换使用)
        mov al,bl
        push bx
        mov bx,0000h            ;DS:BX 为传送数据地址
        mov cx,0078h            ;写入扇区数 60K/512
        int 26h                 ;A 缓冲区的 60K 数据存入磁盘
        pop ax
        mov bx,ax
        add dx,0078h            ;逻辑扇区号曾加 78H
        cmp dx,02d0h
        jnz loop1               ;未满 360K 则下次仍存入该盘,否则换驱动器
```

```
        inc al
        and al,01h
        mov bl,al
        mov ah,01h
        int 16h                    ;如有键按下则采集系统停止工作,否则继续
        jnz stop
        mov ax,bx
        jmp loop0
stop:   mov dx,268h
        out dx,al                  ;停止采集系统工作
        int 20h
        ret
start   endp
code    ends
        end begin
```

4.3 步进电机接口

在工业控制系统中,通常要控制机械部件的平移和转动。这些机械部件的驱动大都采用交流电机、直流电机和步进电机。在这 3 种电机中,步进电机最适合于数字控制,因此它在数控机床等设备中得到广泛的应用。在设计微机工业控制系统时,如何设计步进电机接口,这是一个经常要遇到的问题。

4.3.1 步进电机的基本工作原理

步进电机,顾名思义,它是一步一步转动的。例如每一步,电机轴转动 1°,它可以顺时针转,也可逆时针转。因此,其功能完全和一般电动机相似。

一般步进电机控制电路框图如图 4.5。它由脉冲分配电路和驱动电路构成。脉冲分配器

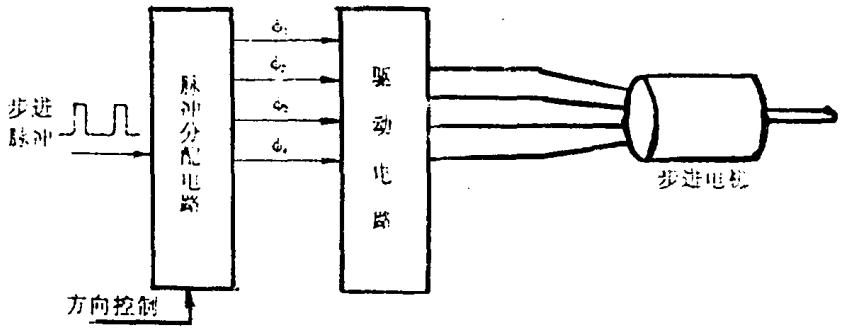


图 4.5 步进电机控制电路框图

有 2 个输入信号:一个是步进脉冲。每输入一个步进脉冲,脉冲分配器的 4 相输出时序将发生一次变化,从而使步进电机转动一步。另一个是方向控制信号。它的 2 个不同状态将使脉冲分配器产生不同方向的步进时序脉冲,从而控制步进电机顺时针转动还是逆时针转动。脉

冲分配器的 4 相激磁信号经驱动电路后，再接到步进电机的激磁绕组上，对步进电机进行功率驱动。

4.3.2 脉冲分配器及驱动放大电路

一、脉冲分配器

如前所述，脉冲分配器的任务是在步进脉冲的激励下，产生 4 相相应的步进激励脉冲。实际上是一个时序产生电路，其构成的方法很多。

随着大规模集成电路技术的发展，现在已生产出专门用于步进电机控制的脉冲分配器芯片，它可以适用于 3 相和 4 相步进电机的各种激励方式，TD62803P 就是其中一例。TD62803P 的引脚图如图 4.6 所示，其各引脚定义如下：

CW/CCW：正转/反转控制

E_A、E_B：激磁方式控制

3/4：3 相或 4 相切换控制

$\overline{MO}$：初始状态检出，初始状态时其输出为低电平

ϕ_1、ϕ_2、ϕ_3、ϕ_4：4 相驱动脉冲输出

E：输出允许，当该端为高电平时，允许 ϕ_1～ϕ_4 输出

CKOUT：时钟输出，它可以用来对步进脉冲进行计数

CK1、CK2：时钟输入

$\overline{R}$：复位输入

GND：地

V_{CC}：+5V 电源

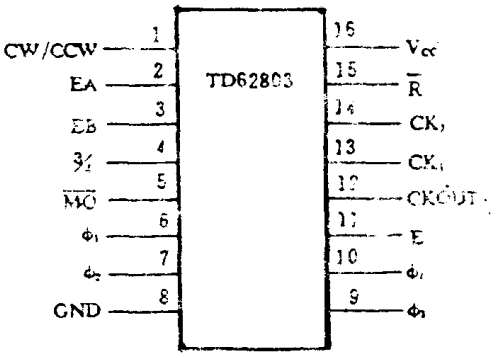


图 4.6 脉冲分配器 TD62803P 引脚

从 TD62803P 的引脚定义可以看到，它是一个功能很强的功能可控的多功能脉冲分配器。在相应引脚上加上不同的控制电平即可得到不同的控制功能。有关控制功能的真值表如表 4-1 所示。用 TD62803P 和有关接口芯片相连接就很容易构成一个用微型计算机控制的步进电机接口电路。

表 4-1 TD62803P 控制真值表

CK_1	CK_2	CW/CCW	功能
⌐	H	L	CW
⌐¬	L	L	禁止
H	⌐	L	CCW
L	¬⌐	L	禁止
⌐	H	H	CCW
⌐¬	L	H	禁止
H	⌐	H	CW
L	¬⌐	H	禁止

(a)

E_A	E_B	3/4	功　能
L	L	L	4 相 1 相激磁
H	L	L	4 相 2 相激磁
L	H	L	4 相 1—2 相激磁
H	H	L	测试模式，输出全部有效
L	L	H	3 相 1 相激磁
H	L	H	3 相 2 相激磁
L	H	H	3 相 1—2 相激磁
H	H	H	测试模式，输出全部有效

(b)

二、驱动放大电路

一般脉冲分配器输出驱动能力是有限的，它不可能去直接驱动步进电机，而需要经过一级功率放大以后，再去驱动步进电机。最简单的方法是再经一级晶体管功率放大电路去推动步进电机。如果步进电机功率不太大的话，目前市场上已有集成功率放大芯片出售，例如，TD62308AP，它可以与 TD62803P 配合，驱动电源电压小于 50V，电流小于 1.25A 的步进电机。在使用大功率步进电机时，也用它来进行预置功率驱动。

4.3.3　步进电机控制接口实例

例 4.3　步进电机接口

图 4.7 是一个实用的步进电机控制接口实例。它包括一个并行输出接口，一个 D/A 输出接口和一个定时器及相应的步进电机控制和驱动电路。

从图中可以看到，微型计算机通过 D/A 接口将一个模拟电压加到压控振荡器(74LS624)的电压控制输入端，一定频率的步进脉冲从 CK_1 输入，利用 D/A 输出电压的高低，可以控制压控振荡器的频率，也就是说可以控制步进电机的转动速度。并行接口的两个输出端分别控制压控振荡器的启停和脉冲分配器的正/反转。脉冲分配器的 CKOUT 输出接到微型计算机的某一个定时通道，该定时器用来计数步进电机的步数。

现在我们来看一下步进电机的工作控制过程。现在假设步进电机要前进(正转)100 步。那么先向定时器置一个 100 的数，并且允许计数到"0"时产生中断。再向并行端口的 CW/CCW 端送一个高电平，下面使步进电机处于正转状态。在初始状态下，压控振荡器启停控制端应为高电平，禁止压控振荡器工作。接着给 D/A 接口送一个让步进电机按某一速度转动的一个数。压控振荡器可按某一频率送出步进脉冲。一切准备就绪后，向并行端口 STOP 端送一个低电压，压控振荡器开始工作，以某一频率输出步进脉冲，步进电机以某一速度正向转动。每转动一步 CKOUT 输出一个步进脉冲，使定时器的计数值减 1。当 100 步走完时，定时器产生中断，CPU 向并行端口 STOP 端送一个高电压，压控振荡器停止工作，步进电机停止转动。

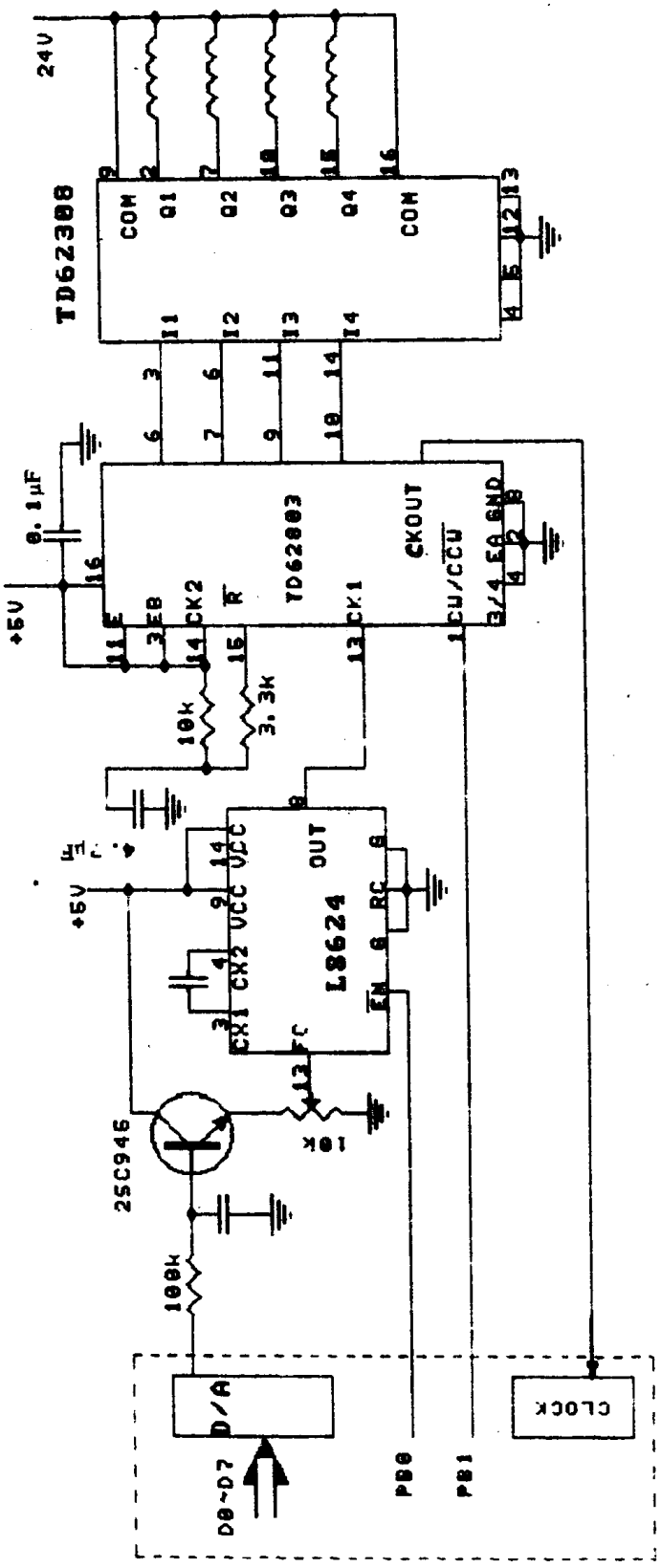


图 4.7　步进电机控制接口电路

4.4 多媒体系统中的接口

4.4.1 多媒体系统的基本构成

一、迅速发展的多媒体技术

多媒体系统是对各类信息(如文字、声音、图形、图象等)进行存储、加工、显示、播放、传输、交换以及供人们直接交互使用的计算机应用系统。九十年代计算机技术一个重要的发展方向就是多媒体技术,它改变了传统计算机只能单纯处理数据和文字信息的不足,使计算机能够综合处理声音、文字、图形和图象等信息,并以形象、丰富和方便的交互性,极大地改善了人机界面,改变了使用计算机的方式,从而为使计算机进入人类生活和生产的各个领域打开了大门,它为计算机产业开辟了非常广阔的市场。

与多媒体有关的软硬件产品,1993 年迈入增长期之后,1994 年至现今呈飞速增长趋势。

(1)全球个人电脑去年达 1.5 亿台。其中多媒体电脑去年增加 850 万台达 4560 万台,预计今年将再增加 1 千万台。

(2)只读光盘机(CD-ROM 驱动器)。仅 1994 年就销售 1745 万台,预计 1995 年销售 2450 万台,1996 年销售 3180 万台,1997 年销售 3800 万台。

(3)音效卡。去年销售 1694 万片,预计今年将销售 1930 万片,96 年销售 2150 万片,97 年销售 1940 万片。

(4)视频压缩/解压缩芯片,1994 年销售 183.3 万套,预计 1995 年将达 643.4 万套,96 年达 1382.6 万套,97 年达 2355 万套。

多媒体 PC 机领导着家用电脑的新潮流。去年个人电脑以难以遏制的势头迅速进入了家庭和教育市场。个人电脑将首先以其娱乐和家庭教育两项功能赢得千家万户的青睐。多媒体技术则是其魅力的关键所在。

目前世界上很多国际性的大公司都在研制开发多媒体计算机技术,其中比较典型的多媒体系统有:

(1)PhiLips/Sony 公司的 CD-I 系统

PhiLips/Sony 公司于 1986 年 4 月公布了基本的 CD-I 系统,同时还公布了 CD-ROM 的文件格式,这就是以后的 ISO 标准,该系统把高质量的声音、文字、程序、图形、动画/静止图像等都以数字的形式存放在容量为 650MB 的 5 英寸只读光盘上。用户可通过与该系统相连的家用电视机、计算机显示器和 CD-I 系统进行通信,使用鼠标器、遥控器等定位装置选择感兴趣的视听材料进行播放。

(2)Intel 和 IBM 公司的 DVI 系统

1989 年,Intel 公司与 IBM 公司推出了 DVI(Digital Video Interactive)系统的第一代产品 Action Media 750 I 。1991 年又推出了 DVI 系统的第二代产品 Action Media 750 I 。

DVI 用户系统是以 IBM-PC/AT、386、486 或其兼容机作工作平台,同时配有 CD-ROM 驱动器,带有音响效果的 RGB 彩色显示器以及键盘,并配置了三个专用的 DVI 接口卡,即 DVI 视频卡、DVI 音响卡以及 DVI 多功能接口卡。

DVI 开发系统是在用户系统的基础上再配备与多媒体有关的视频信号数字化器(连接到 DVI 视频卡上)、音响数字化器(连接到音响卡上)、扩展的视频 RAM、大容量光盘或硬盘、磁带机、摄录象机、音响、监视器和扫描仪等外部设备。

(3)Apple 公司的 Macintosh 系统。

Apple 公司的 Macintosh 系统具有公认的良好的图形特性,它是桌上出版和桌上展示系统的先驱。该公司最近向中国广大用户推出了其家用电脑中最经济实惠的 Macintosh LC630 型机器。它不仅综合了多媒体的全部功能(16 位立体声输入输出,CD-ROM 驱动器,AV 及 S-Video 端子,以及 TV 调谐器等)外,还具有以太网络接口功能。它的多媒体扩展选件可为用户提供崭新的信息交流方法。比如可通过加设选件将电脑连接到大屏幕电脑来显示图象,通过加设内部 TV 调谐器,在显示器的视窗内直接观看电视节目。

二、多媒体系统的基本组成

多媒体系统没有固定的配置模式,通常多媒体硬件系统应包括如下一些部件:

(1)主机。通常选用 386 系列 25M 主频以上主机,2M 以上内存,120M 以上硬盘的机型。对于大量处理声音、图象、动画或视频的应用需要较大的内存和硬盘;一些多媒体产品对硬件环境也有特殊的要求,如视频卡需要 VGA 提供特征口。

目前多媒体系统的主机主要包括 Apple 公司系列微机和 PC 系列微机及兼容机,其他还包括 SUN、DEC 等型号的工作站。

(2)音频部件。声卡可以完成声音采样及播放,语音合成卡可将正文合成语音,再配有 CD-ROM、话筒和音箱、录音机就可以构成一个完整的音频环境。

目前世界上销量最大的多媒体板级产品是声卡,在美国年销售量已达 230 万块,最流行的是 Creative Labs 公司的 Sound Blaster 16 系列卡(声霸卡)和 Sound Man 16 系列卡。

CD-ROM 驱动器是多媒体 PC 的心脏和灵魂。对一般用户,可选用 SONY CDU-33A (SONY 接口)和 SONY CDU-55E(IDE 接口)以及 Mitsumi 驱动器。这三种驱动器性能居中,价格便宜,支持的 CD 标准广泛。

(3)视频部件。视频卡可以完成视频的播放和采集,视频编码卡可将 VGA 信号编码为标准视频信号输出,再配上摄象机、录相机、电视机即可构成一个完整的视频处理系统。动态视频压缩卡可以完成视频的采集及动态压缩,可将视频实时压缩存入硬盘并回放。静态图象压缩卡作为图象处理系统的选件,可以完成图象的压缩。

目前市面上的视频卡种类繁多,且互相兼容性差,质量和价格均相差很大。常见的有:Miro Video DC1 和 Motion Star Pro 视频捕捉卡,Reelmagic、Ledlead 110 和 V30 回放卡。

三、多媒体软件系统

多媒体软件系统应包括以下几个部分:

(1)多媒体计算机系统软件

多媒体计算机系统软件是多媒体计算机软件产业的底层软件,它包括多媒体操作系统和多媒体 Windows。Intel 和 IBM 公司开发的 AVSS 和 AVK(Audio Video Subsystem & Audio Video Kernal)就是在 DOS 和 Windows 支撑环境上,增加些 Audio、Video、CD-ROM 等驱动程序,同时,为了适应多媒体数据实时性的要求,扩充了 DOS 的实时性。Phillips 和 Sony 公司为

CD-I 系统研制的 CD-RTOS(Compact Disc Real Time Operating System),这种光盘实时操作系统,支持多种数据类型和数据流,即视频、音频、文字、控制和应用程序,还支撑多种 I/O 设备、多个数据流的多个进程,需要同步或异步,而且要在限定时间内完成。另一种比较通用的方案是在现有的 DOS 和 Windows 条件下,加上多媒体板级产品的驱动软件和 Demo 软件以及开发工具软件(Development Kit)。

(2)多媒体数据库和编辑工具

多媒体数据库是建立多媒体信息管理系统、多媒体应用系统最重要的工具,它需要建立多媒体数据统一的数据格式,多媒体数据的存储模型,多媒体数据的检索模型以及多媒体数据的管理模型。多媒体编辑工具的设计目的是缩短多媒体应用软件的研制开发时间,降低对制作人员素质的要求,多媒体编辑工具可分成下述三类:高档编辑工具:适合电影、电视系统专业编辑工具;中档编辑工具:适合教材、娱乐系统的制作编辑;低档编辑工具:适用于商业介绍资料、简报及家庭学习材料的编辑。

目前世界上比较流行的,在 Windows 平台上运行的编辑工具有:Macromedia 公司的 Authorware Professional,它具有突出的交互式功能;Aimech 公司的 Icon Author,它有 50 多个图符(Icon),用流程图方式建立应用程序;Asymetrix 公司推出的 Multimedia Tool Book,它的特点是按书的结构组织应用程序,具有较强的超级文本(Hypertext)超级连接能力;Macro Media 公司推出的 Action! 它的基本组成是节目(Presentation)、场景(Scene)和对象(Object),它还有四种工具(Tool)和三个窗口。清华大学计算机系开发的 Avstar 多媒体编辑工具,是在 DOS 环境下设计的,两个主要组成部分是:Mcmd 命令解释器和结构编辑器。现在他们正在设计 Avstar for Windows 它的特点是:开放式体系结构;内嵌式多媒体数据库;引入时间轴概念处理媒体间同步问题以及采用基于工具箱的交互式操作方式。台湾九二八电脑公司的洪图编辑系统,可以用在电脑辅助教学(CAI)和电子图形的开发制作,它由编辑系统、执行系统和辅助工具三大部分组成,同时提供了与其他高级语言的接口及简单的描述语言。

(3)多媒体应用系统

利用多媒体数据库和多媒体编辑工具,可以方便、迅速地编制出极有效益的多媒体应用系统:多媒体办公自动化系统;多媒体工程数据库系统;多媒体人事档案管理系统;多媒体地理管理系统;各种电子出版物。

4.4.2 多媒体系统的接口实例

例 4.4 I 型 DVI 系统接口

DVI 技术第一代的产品 Action Media 750 I 的结构如图 4.8 所示,其核心是三块专用的 DVI 接口板:DVI 视频板、DVI 音频板和 DVI 多功能板。IBM PC/AT,386,486 或其兼容型计算机作为工作平台,同时配有 CD-ROM 驱动器,带有放大器和音响效果的 RGB 彩色监视器,组成了 DVI 用户系统。在此基础上再配置与多媒体有关的外设,视频信息数字化器(连接到 DVI 视频板上),音响信号数字化器(连接到 DVI 音响板上),扩展的视频 RAM(连接到视频板上),大容量的光盘或硬盘、磁带机、录象机、音响设备、监视器以及摄象机或扫描仪等即可组成 DVI 开发系统。

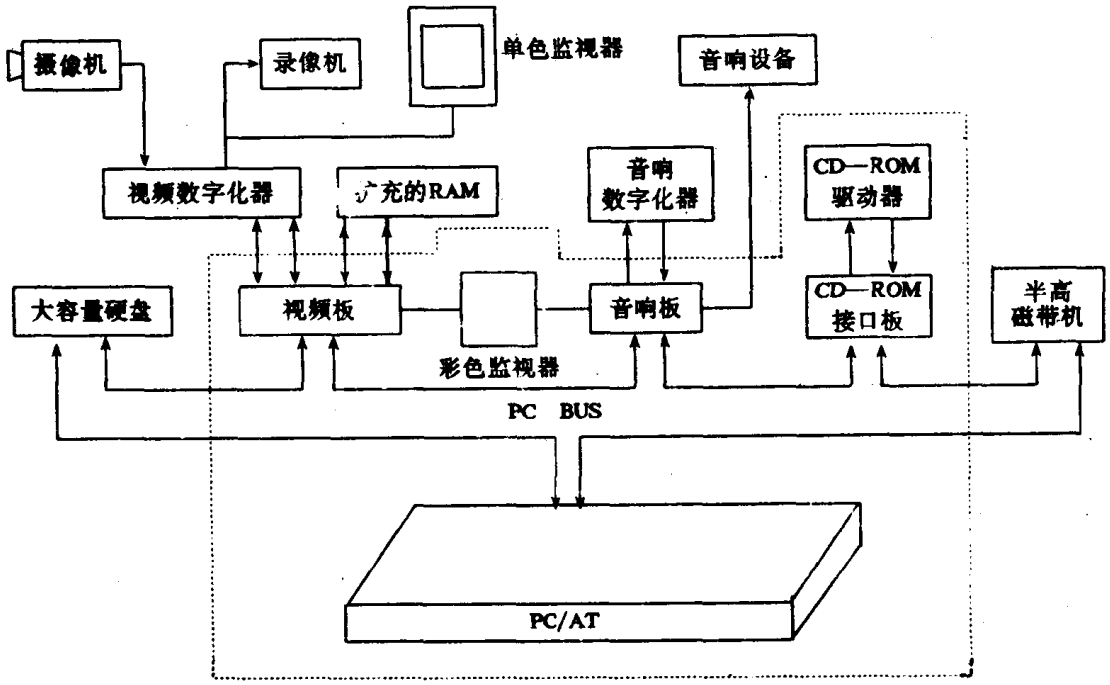


图 4.8 DVI 系统结构图

一、DVI 视频接口板

DVI 视频接口板的原理方框图如图 4.9 所示。DVI 视频接口板的核心部件是 Intel 公司生产的专用芯片：VDP1(82750 PA)象素处理器以及 VDP2(82750 DA)显示处理器。82750 PA 象素处理器，象素操作速度为 12.5M/s，它采用微码编程，可以高速执行象素处理的多种算法。82750 DA 是显示处理器，它可与 82750 PA 并行操作。当视频象素处理器绘制和管理视频 RAM 中的位映射图时，显示处理器就把这个结果显示在视频屏幕上。82750 DA 象素处理器可以通过编程控制寄存器，适应不同分辨率，不同象素格式及不同同步格式的多种型号的显示器。

在图的右上角是 VRAM，由 256KB VRAM 芯片构成 1MB 的模块，最多可安装四个模块，VRAM 的容量为 4MB。如果使用 1MB VRAM 芯片，最大容量可达 16MB，这也是 82750 PA 可寻址的最大容量。象素处理器 82750 PA 到 VRAM 采用 32 位并行的数据通道，一个 VRAM 读写周期存取一个 32 位字长的象素，如果象素字长较短时，一个 VRAM 读写周期可存取多个象素。

在主机和 VRAM 之间，有一个绕过 82750 PA 单独的通道接口，它受 82750 PA 的仲裁。由于主机的数据通道字长是 16bit，所以主机一次只能存取 16 位 VRAM 数据。为了能够寻址 16MB VRAM 地址空间，把它分成 256 个字段，每段容量 64KB，它可通过硬件跨接线设置初始地址，直接连到 IBM PC/AT 的存储空间，选取不同的段可通过软件实现。

为了控制 82750 PA 的工作状态，主机可通过设置 82750 PA 象素处理器中 256 个 16 位字长的寄存器的内容来实现。82750 PA 是一个微处理器，具有一套单独的指令系统，为了运

行这些指令需要对它专门编程。需要 82750 PA 运行时，执行的是存在 82750 PA 芯片 RAM 中的微码，执行前主机将 82750 PA 的微码先加载到 VRAM 中，然后由 82750 PA 再把它从 VRAM 中传送到自己芯片的 RAM 中，以便提高运行速度。

82750 DA 显示处理器没有直接连到 PC 系统总线上，它连接到 82750 PA 象素处理器和 VRAM 的串行通道上。为了满足彩色显示器高速数据传送的需要，把 VRAM 输出到 82750 DA 串行通道的字长设计成 2×32 位，再经过多路开关送给 82750 DA 显示处理器，变成标准的 32 位象素编码的形式。

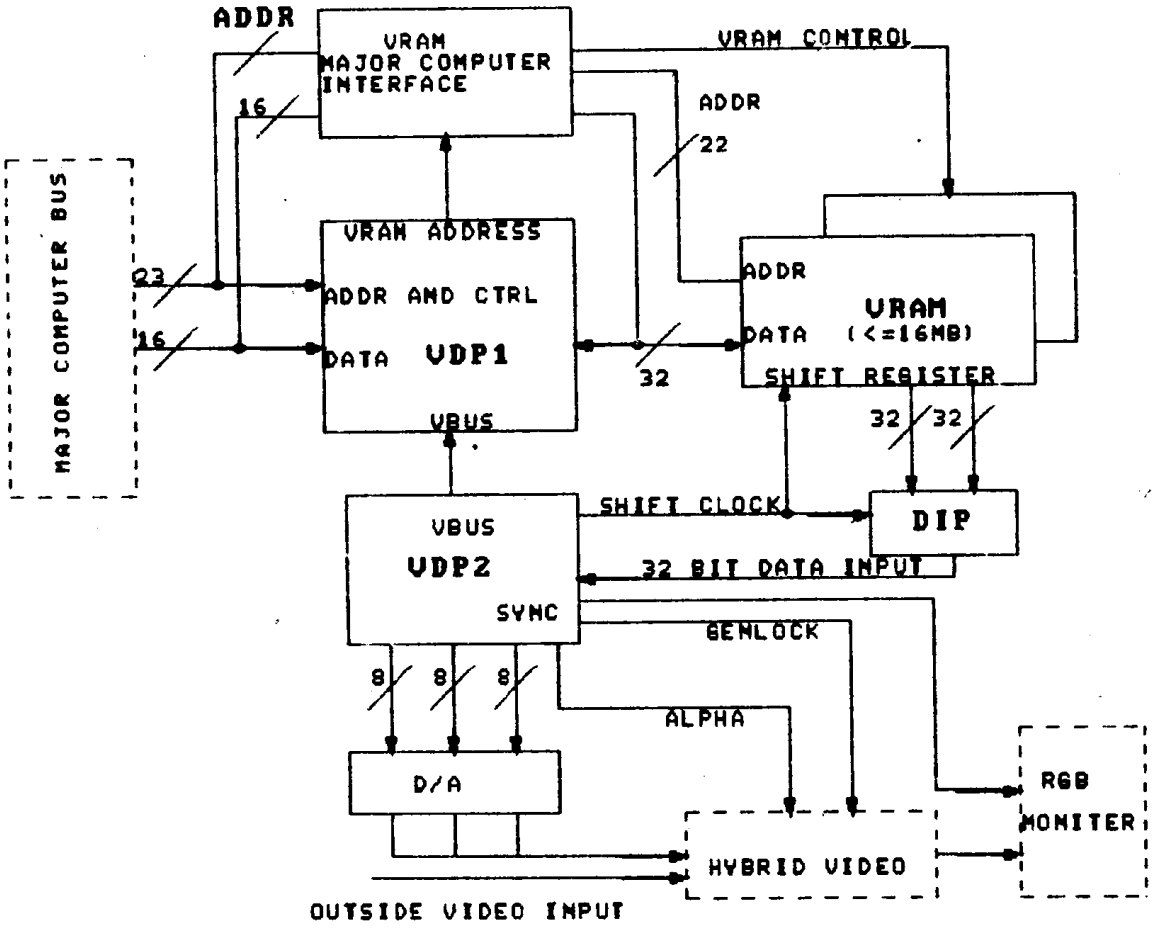


图 4.9 DVI 视频接口板原理框图

二、DVI 音频接口板

DVI 音频接口板的原理如图 4.10 所示。TMS-320C10 是 TI 公司生产的数字信号处理器(DSP)，在这里做为专用的音响处理器，它使用 4096×16 位高速静态 RAM 存放音响处理程序，称为程序存储器。

音响数据存储器是由 16KB(8K Word)的动态 RAM 组成，通过 I/O 通道与 TMS-320C10 相连，同时，用同样的方式与 PC 总线相连，因此，主机的 CPU 和 TMS-320C10 共享该存储器。音响数据存储器可做为音响数据的缓冲区。从硬盘或 CD-ROM 存取的压缩编码的音响数据，都要通过它再送到音响专用处理器。最多它能够存储 4 秒钟经过压缩编码的音响信号，它的最大贡献是有效地利用了 PC 总线的带宽。

经过 TMS-320C10 音响协处理器输出通道得到的数字式音响输出信号，送到两个 14 位

D/A 变换器转换成模拟信号，再送到两个可编程带宽的模拟滤波器进行滤波。左右两个通道输出可产生较好的立体声音响效果。当然两个通道还可由软件编程做为其他用处。

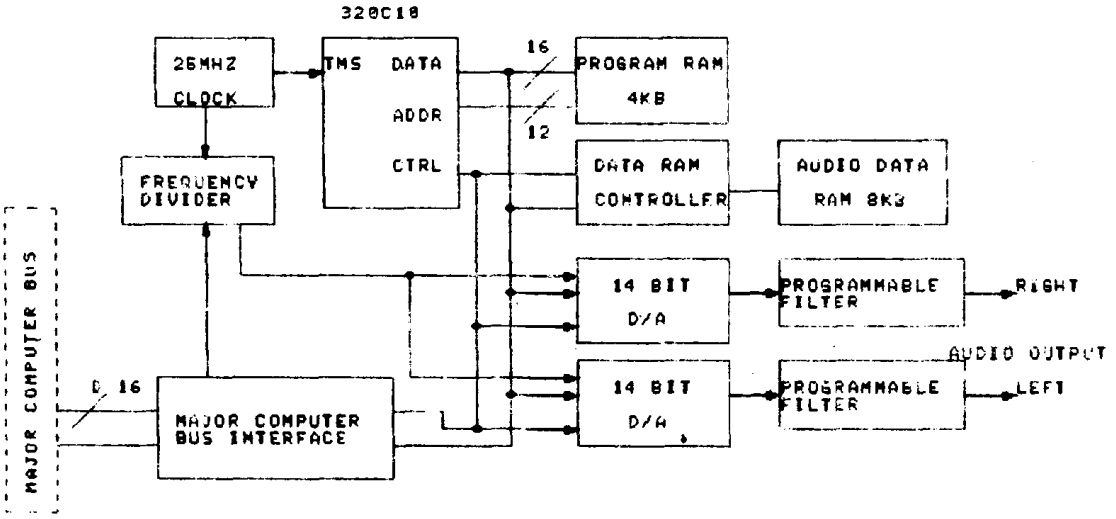


图 4.10 DVI 音频接口板原理框图

三、DVI 多功能接口板

DVI 多功能接口板原理如图 4.11 所示，它由三个功能块组成：

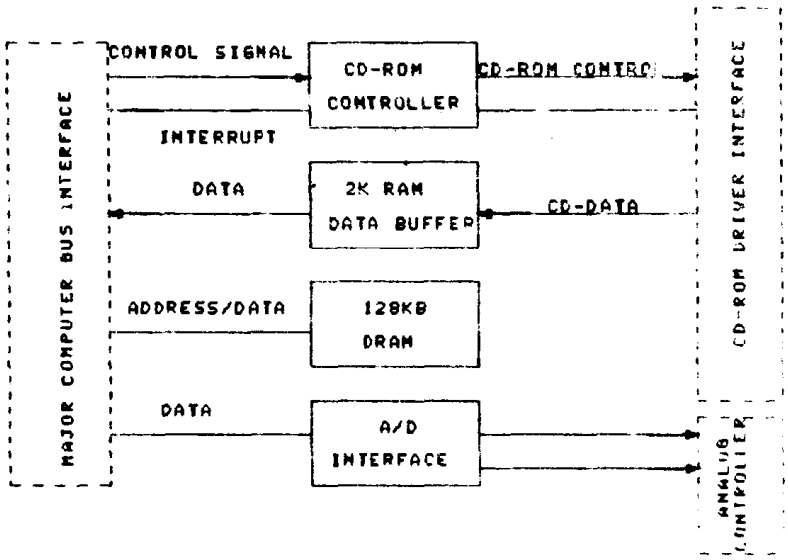


图 4.11 DVI 多功能接口板原理框图

CD-ROM 接口控制器：I 型 DVI 系统 CD-ROM 控制器是为 Sony CDU-100B 型独立的 CD-ROM 驱动器而设计的，它也适用于其兼容产品。DVI 系统一个 CD-ROM 盘片能够存放 72 分钟全运动全屏幕的视频图象或 17 小时的讲话信息和 5000 幅静止图象。CD-ROM 控制

器由 CD-ROM 控制逻辑和 2KB RAM 数据缓冲器组成。

扩展内存模块：为了支持内存容量为 512KB 和 IBM PC 母板，扩充到 DOS 支持的 640KB，内存扩充模块中 DRAM 的容量是 128KB。如果不需要，可通过软件把它屏蔽掉。

两路操纵杆控制器接口：它用 A/D 变换器把操纵杆的位置变成数字码送给主机，数字码的转换时间为 10ms。

第五章　微机通信接口技术

5.1　异步串行通信接口

5.1.1　异步串行通信接口标准

EIA-RS-232C 标准是由美国电子工业协会(EIA)从 CCITTV・24 建议派生出来的用于串行通信的标准。用于数据终端设备(DTE)和数据通信设备(DCE)之间串行二进制数据通信。

一、接口的信号定义

RS-232C 标准接口共有 25 条引线，这 25 条引线的信号定义如表 5-1。RS-232C 接口所定义的信号线较多，但实际上有许多是很少使用的。其中，发送数据(TXD)、接收数据(RXD)、信号地(SG)，这三条是最基本的。数据准备好(DSR)、数据终端准备就绪(DTR)、接收线路信号检测(DCD)和音响指示(RI)是针对电话网络设计的。在本地互连的微机系统中，最常用到的联络信号 DTR、DSR、RTS、CTS。

表 5-1　RS-232C 标准引脚信号

引脚	含　义	引脚	含　义
1	保护地	14	第二通道发送数据
2	发送数据(TXD)	15	传输信号单元定时
3	接收数据(RXD)	16	第二通道接收数据
4	请求发送(RTS)	17	接收信号单元定时
5	清除发送(CTS)	18	未分配
6	数据准备好(DSR)	19	第二通道请求发送
7	信号地(SG)	20	数据终端准备就绪(DTR)
8	接收线路信号检测(DCD)	21	信号质量检测
9	接收线路建立测试	22	音响指示(RI)
10	接收线路建立测试	23	数据信号速率选择(DSRD)
11	未分配	24	发送信号单元定时
12	第二通道接收线信号检测	25	未分配
13	第二通道清除发送		

二、接口的电气特性

在 RS-232C 中任一条信号线的电压为负逻辑关系，即逻辑“1”：－5V～－15V；逻辑“0”：＋5V～＋15V。噪声容限为±2V。

三、接口的机械特性

RS-232C 采用 DB-25(即 D 型 25 脚连接器)作为接口连接器。DB-25 符合国际标准化组织制定的 ISO 2113 标准。DB-25 的机械结构图如图 5.1 所示。

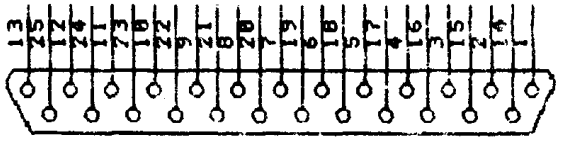


图 5.1 DB-25 连接器机械结构

5.1.2 串行通信接口的硬件连接

利用 PC 机的 RS-232C 串行接口总线可以实现 PC 机与其他具有 RS-232C 接口的设备或计算机的连接。典型的连接方式如下。

1. 全双工标准连接(DTE-DCE),例如计算机-调制解调器之间(图 5.2)。
2. 三线连接(DTE-DTE),例如计算机-计算机之间(图 5.3)。

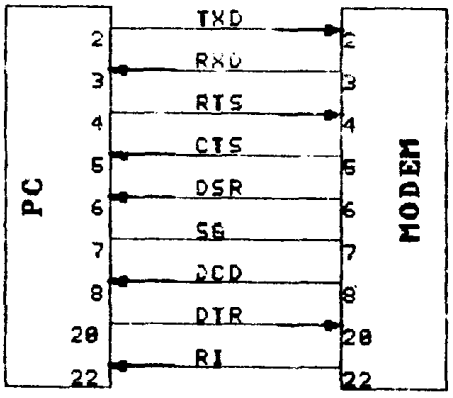


图 5.2 PC 机与 MODEM 连接

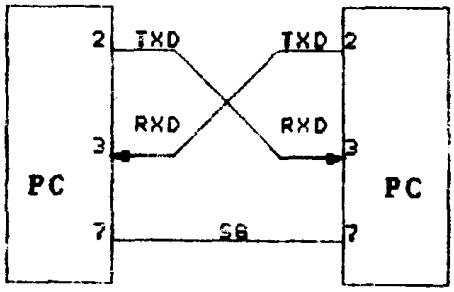


图 5.3 三线连接

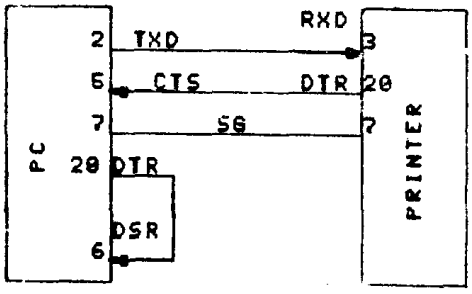


图 5.4 反馈连接

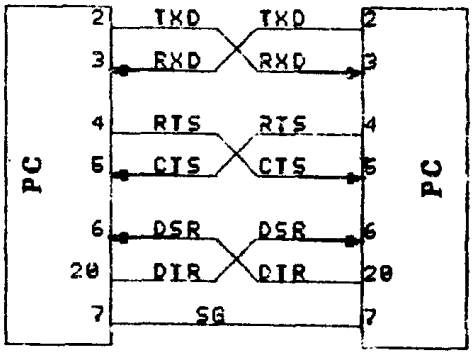


图 5.5 PC 系列微机之间互连

3. 反馈连接(DTE-DTE),例如计算机-打印机之间(图 5.4)。

4. 交叉连接(DTE-DTE),例如同一型号的计算机之间(图 5.5)。

5.1.3 串行通信接口方法

PC 系列微机通信接口有两种方法:一种是 BIOS 调用和 DOS 系统调用,另一种是直接面向 8250 的通信接口方法。而后一种方法要灵活得多,它可以用查询方式进行通信,也可以用中断方式进行通信。

一、通信接口的 BIOS 和 DOS 调用方法

1. 通信接口的 BIOS 调用

通信接口的 BIOS 调用 INT 14H 有 4 个功能号 0~3,其功能如下:

(1)初始化通信口(0 号功能)

调用参数:AH=0,BX=0(若安装了一个以上通信适配器,BX 可为 0 或 1),AL=初始化参数。

返回参数:AX 中的内容为线路状态寄存器和调制解调器状态寄存器中的内容。

这种功能调用不允许通信带有中断。

(2)发送数据字符(1 号功能)

调用参数:AH=1,BX=0,AL=要发送的字符

返回参数:AH 中的内容为线路状态寄存器的内容。若字符没有发送,AH 的位 7 置 1。

(3)等待接收字符(2 号功能)

调用参数:AH=2,BX=0

返回参数:AL=接收的字符;AH=线路状态寄存器内容,但位 5,6 为零。

(4)读通信口状态(3 号功能)

调用参数:AH=3,BX=0

返回参数:AH=线路状态寄存器内容,AL=调制解调器状态寄存器的内容。

2. 通信接口的 DOS 系统调用

(1)异步通信口输入(3 号调用)

从异步通信口等待输入一个字符,存入寄存器 AL 返回。注意:该通信口在 DOS 启动时,8250 已初始化为 2400 波特、一个停止位、无奇偶校验、8 位数据位;且不返回状态和错误码。

(2)异步通信口输出(4 号调用)

把寄存器 DL 中的字符输出到异步通信口。它与功能调用号 3 一样,也不返回状态和错误码。

二、直接面向 INS 8250 的通信接口方法

PC 系列微机的异步通信适配器是以芯片 INS 8250 为中心构成的,直接面向 INS 8250 的通信接口方法如下:

1. 8250 的初始化

8250 的初始化包括设定数据传输率,为线路控制寄存器、MODEM 控制寄存器和中断允许寄存器等设定初值。

(1)设置波特率

首先应把线路控制寄存器的最高位置1(端口地址3FB),然后把给定的波特率所对应的除数值的低位字节与高位字节分别从端口地址3F8和3F9送出。这样就设定了数据传输率。

(2)设置通信数据格式

根据数据格式(字符长度、停止位数目、奇偶校验方式等)设定线路控制寄存器的数值(通常较高的3位设为0),由端口地址3FB送出。

(3)设置MODEM控制寄存器

MODEM控制寄存器一般可设定为03,意欲使用中断时设为0B。

(4)设置操作方式和中断允许寄存器

若不使用中断,则中断允许寄存器应设置为0;否则按需要设定中断允许寄存器。

2. 通信接口方法

(1)通信接口的查询方法

CPU把字符送给8250的发送保持寄存器发到线路上去,从线路上接收到的字符保持在接收缓冲寄存器中。然而只有当发送保持寄存器是空的时候,才能接受待发送的字符;只有当接收缓冲寄存器中确已收到来自线路的字符时,读取这个字符才是有意义的。那么,怎样知道发送保持寄存器空或者接收缓冲寄存器已收到字符呢?正是线路状态寄存器(端口地址3FD)提供了有关信息。例如D5=1表示发送寄存器已空,因此在D5=0时不能把字符送给发送保持器。与此类似,D0=1表示8250已接收到一个字符,并存放在接收缓冲器中。因此必须及时读取接收缓冲器中的字符(应保证在8250接收完下一字符之前读取),否则将发生字符重叠的错误。

线路状态寄存器还可用来检查所接收数据的错误状态或其他线路状态。

MODEM状态寄存器使我们获知来自MODEM的输入信号的状态(第4位~第7位)。最低4位使我们获知,自从上一次读过此寄存器之后,MODEM送来的输入信号是否改变了状态。

(2)通信接口的中断方法

为了提高CPU的工作效率,可以不采用查询方式,即不是通过检查8250线路状态寄存器的状态来传输字符,而是当事件发生时对主程序进行中断,然后转入相应的中断服务程序去处理发送数据或接收数据。

当中断发生时,中断识别寄存器的内容可告知产生中断的原因。当数个中断条件同时发生时,8250自动按其规定的优先级别予以处理。故一个中断服务完毕时应再度检查中断识别寄存器,看是否还有别的中断请求等待服务。

8088CPU不允许多重中断,即中断服务程序不可能再被中断。如果我们只许"接收数据就绪"一种中断,则当中断发生时,就用不着检查中断识别寄存器的内容。此时中断服务程序应向接收缓冲器读取数据,且检查线路状态寄存器,看接收的数据是否有错误。

5.1.4 串行通信接口实例

例5.1 以查询方式发送文件

要实现两台PC机之间的近距离通信,其硬件接口按图5.5所示连接,软件接口用查询方式。

首先对8250进行初始化,设置波特率;经对线路控制寄存器设置数据传送格式;再设置

MODEM 寄存器和中断控制寄存器。然后，建立一个字节的磁盘传输区 DTA。即通过 INT 21H 的 1AH 功能调用号，设置 PSP 程序前缀控制块 PSP+80H。通过 SETFCB 的过程，从键盘输入文件标识符(包括磁盘驱动器号、文件名和扩展名)分别填入文件控制块 FCB 中(即设置 FCB)。接下来要打开用户指定文件，顺序读。若读文件有错，就显示错误信息返回：若读文件无错，则按查询方式发送字符。在查询中，先检查线路寄存器，若为空，即可将刚从文件中读入的字符发送出去；若不空，就查询等待。这样重复发送字符，直至结束字符 CTRL-Z (即 ASCII 码 1AH)，就关闭文件返回。程序清单如下：

```
;       exam51.asm

stack   segment para stack ' stack'
        db 256 dup(0)
stack   ends
data    segment para public ' data'
fcb     db 37 dup(0)
dta     db 0
count   db 0
inmsg   db ' input filename:'
        db 10,13,' $'
errmsg  db ' file access error! '
        db 10,13,' $'
err     db ' input error! $'
data    ends
code    segment para public ' code'
start   proc far
        assume cs:code,ds:data
        assume es:data,ss:stack
        push ds
        mov ax,0
        push ax
        mov ax,data
        mov es,ax
        mov ds,ax
        ;8250 初始化
        mov dx,3fbh                    ;置 DLAB=1
        mov al,80h
        out dx,al
        mov dx,3f8h                    ;置波特率因子低位(600BIT/s)
        mov al,0c0h
        out dx,al
        mov dx,3f9h                    ;置波特率因子高位
        mov al,0
        out dx,al
```

```
        mov dx,3fbh                 ;置数据格式为 7 位数据位,1 位停止
                                    ;位,奇校验
        mov ax,0ah
        out dx,al
        mov dx,3fch                 ;置 MODEM 控制寄存器,关中断
        mov al,03h
        out dx,al
        mov dx,3f9h                 ;屏蔽中断
        mov al,0
        out dx,al
        ;建立 DTA
        mov dx,offset dta
        mov ah,1ah
        int 21h
        ;接收键盘输入文件名,并填写 FCB
ple:    call setfcb                 ;设置 FCB
        ;打开指定文件
        mov dx,offset fcb
        mov ah,0fh
        int 21h
        cmp al,0
        jnz ple
        mov word ptr fcb+0ch,0      ;置当前块号为 0
        mov word ptr fcb+0eh,1      ;置记录长度为 1
        mov fcb+20h,0               ;置当前记录号为 0
again:  mov dx,offset fcb           ;顺序读文件(一个字符)
        mov ah,14h
        int 21h
        cmp al,0
        jnz error
send:   mov dx,3fdh                 ;查询线路状态寄存器的位 5,若发
                                    ;送保持寄存器空,发送一个字符
        in al,dx
        test al,20h
        jz send
        mov al,dta
        mov dx,3f8h
        out dx,al
        cmp al,1ah                  ;若是文件结束符(CTRL-Z)
        jz eof                      ;转 EOF
        call dispchar               ;显示字符
        jmp again
error:  mov dx,offset errmsg        ;错误提示
```

```
        mov ah,9
        int 21h
        ret
eof:    mov dx,offset fcb          ;关闭文件
        mov ah,10h
        int 21h
        ret
start   endp
setfcb  proc near                  ;SETFCB 过程
sta:    mov dx,offset inmsg        ;显示提示信息
        mov ah,9
        int 21h
        mov ah,1                   ;接收键盘输入的磁盘驱动器号
        int 21h
        cmp al,' A'
        jz x1
        cmp al,' B'
        jz x1
        jmp test1
x1:     sub al,40h                 ;将 A,B 转换为驱动器号 0,1
        mov fcb+00H,AL             ;填驱动器号到 FCB 中
        jmp iner
test1:  cmp al,' a'
        jz x2
        cmp al,' b'
        jz x2
error1: mov dx,offset err          ;输入非 A. B. a. b. 之一,显示错误信息,
                                   ;然后重新输入
        mov ah,9
        int 21h
        jmp sta
x2:     sub al,60h                 ;将 a,b 转换成驱动器号 0,1
        mov fcb+00h,al             ;将驱动器号填入到 FCB 中
iner:   mov ah,1                   ;输入":"
        int 21h
        cmp al,' :'
        jz error1
        mov count,1
        mov di,offset fcb+1        ;DI 指向 FCB 文件各单元
lop:    mov ah,1                   ;接收键入文件名,直到遇到'.'表示
                                   ;文件名结束
        int 21h
        cmp al,' .'
```

```
            jz set1
            cmp count,9                    ;文件名字符个数要小于 9
            jz error1
            stosb                          ;存文件名到 FCB
            inc count
            jmp lop
set1:       mov ah,1                       ;接收键入文件扩展名
            int 21h
            mov fcb+9,al
            mov ah,1
            int 21h
            mov fcb+10,al
            mov ah,1
            int 21h
            mov fcb+11,al
            mov al,0dh
            call dispchar                  ;回车
            mov al,0ah
            call dispchar                  ;换行
            ret
setfcb      endp
            ;显示一个字符子程序
dispchar    proc near
            push bx
            mov bx,0
            mov ah,14
            int 10h
            pop bx
            ret
dispchar    endp
code        ends
            end     start
```

例 5.2 以中断方式实时通信

本程序可驻留在内存，实现 PC 机之间实时通信联络，当连接成功，便可进行通信。程序由四部分组成：定时开放通信口程序，监视通信口接收中断处理程序，键盘中断扩展程序和安装程序。

PC 机的时钟中断处理 8H 包含两项内容：一是计时，另一是管理软盘驱动器的启闭时间。时钟中断脉冲频率为 18.2Hz。在时钟中断处理程序中有一个嵌入软中断 INT 1CH。在 ROM BIOS 中，INT 1CH 的处理程序只是一条空操作返回指令，只要编写一个例行程序，即可实现 1CH 中断服务处理。本程序中就是利用 1CH 软中断，编制定时开放通信口例行程序，使通信口处于经常开放状态。

在 PC 机中，异步通信口的中断请求信号在 DOS 运行后被关闭。这样异步通信口的中断

请求线 IRQ4 所产生的中断请求，无法向系统发出。因此，为了能随时监视异步通信口，达到实时通信目的，在上述 1CH 中断处理程序中，规定了每 10 秒钟打开通信口一次，并设置通信参数。

0CH 通信中断处理程序实现具体监视通信口功能，当发送方要求通信时首先发出呼叫信号。接收方不论处在什么状态，只要通信端口有信号到达，即进入 0CH 中断处理程序进行处理。为简单起见，处理过程只是将光标设置在屏幕底行，通知用户"请准备接收"。如果同意接收，接收方发响应信号通知发送方，至此通信连接即告完成，可以开始运行通信程序，进行通信。

发送呼叫信号和回答信号是 09H 键盘中断扩展程序实现的。这部分程序仅监视 ALT-F10 和 ALT-F9 组合键，对其他所有键码均转入原 INT 09H 程序，保证正常键盘功能。当同时按下 ALT-F10 时，即发出呼叫信号，包括发送 ENQ 和对方站址。同时按下 ALT-F9 时，则发响应信号 ACK。

安装程序实现修改中断向量并使程序常驻内存。1CH 和 0CH 中断向量的修改用 DOS 功能调用 INT 21H 的子功能 25H 填写；09H 中断扩展程序则用 INT 21H 中子功能 35H 读取 09H 原中断向量予以保存，用 25H 填写新的中断入口地址，实现程序的连接。最后用 INT 21H 子功能 31H 将程序常驻内存。

本程序经汇编连接，生成执行文件 exam52.exe 之后，在 DOS 命令状态，键入 exam52，再返回 DOS 时，即安装完毕。此后，只要系统运行，不论处于何种状态，都可起到监视通信口，完成实时通信联络作用。程序清单如下：

```
;     exam52.asm

              ;键盘状态标志单元定义
abso          segment  at 10h
              org 17h
key_flag      db ?                          ;键盘状态标志单元
abso          ends
              ;数据定义
f9            equ 67                        ;F9 键的扫描码
f10           equ 68                        ;F10 键的扫描码
enq           equ 4                         ;ENQ 的 ASCII 码
ack           equ 6                         ;ACK 的 ASCII 码
              ;程序段开始：栈，数据，程序放在一起
code          segment
              assume cs:code,ds:code
count         db 0b4h                       ;10 秒计时值
flag          db 0
bios_int09    dd ?                          ;原键盘中断处理程序入口
mseg          db ' please ready to receive! '
mseger        db ' connection success! '
              ;1CH 软中断处理程序
p_int1c       proc near
```

```
                push ax                         ;保护现场
                push dx
                push ds
                push cs
                pop ds                          ;DS=CS
                mov al,count                    ;十秒计时
                dec al
                mov count,al
                jnz rt                          ;十秒未到返回
                mov count,0b4h                  ;恢复计时初值
                ;10 秒时间到,打开通信口,设置通信参数
                mov al,80h                      ;置 LCR 的 DLAB=1
                mov dx,3fbh
                out dx,al
                mov al,18h                      ;置波特率 4800BPS 除数低字节
                mov dx,3f8h
                out dx,al
                mov al,00h                      ;置除数高字节
                mov dx,3f9h
                out dx,al
                mov al,1bh                      ;置数据格式,数据 8 位,停一位,偶校
                mov dx,3fbh
                out dx,al
                mov al,2ch                      ;置 8259 中断屏蔽寄存器,允许 RS-232
                out 21h,al                      ;串行口中断请求
                mov al,08h                      ;置 MCR 的 OUT2=1,允许送出中断请求
                mov dx,3fch
                out dx,al
                mov al,01h                      ;置 IER 允许数据有效中断
                mov dx,3f9h
                out dx,al
rt:             pop ds                          ;恢复现场
                pop dx
                pop ax
                iret                            ;中断返回
p_int1c         endp
                ;键盘中断扩展程序
p_int09         proc near
                push cx
                push ax
                push ds
                push es
                push cs
```

```
        pop ds                      ;DS=CS
        mov ax,abso
        mov es,ax                   ;ES=40H
        test es:key_flag,8          ;查 ALT 标志
        jnz p2                      ;有 ALT 标志,转
p1:     pop es                      ;无 ALT 标志,去原 INT09 程序
        pop ds
        pop ax
        pop cx
        jmp cs:bios_int09           ;与原 INT09 连接
p2:     in al,60h                   ;读入键码
        cmp al,f10                  ;是 F10?
        jnz p4                      ;不是 F10,去查是 F9?
        mov al,enq                  ;是 F10,发送 ENQ
        call send
        mov cx,100                  ;延时
w:      loop w
        mov al,' c'                 ;发对方站地址
        call send
p3:     in al,61h                   ;读入 8255PB 口内容
        push ax                     ;推入堆栈
        or al,80h                   ;置键盘应答信息
        out 61h,al                  ;送键盘回答信号
        pop ax                      ;恢复 8255PB 口
        out 61h,al
        mov al,20h                  ;发中断结束指令 EOI 给 8259
        out 20h,al
        pop es                      ;恢复现场
        pop ds
        pop ax
        pop cx
        iret                        ;中断返回
p4:     cmp al,f9                   ;是 F9?
        jnz p1                      ;不是,转原 INT09 程序
        mov al,ack                  ;是 F9,发给 ACK
        call send
        jmp p3
        ;发送子程序
send    proc near
        push dx
        push ax
        mov dx,3fdh
        in al,dx
```

⑫

```
                test al,20h
                jz send
                pop ax
                mov dx,3f8h
                out dx,al
                pop dx
                ret
send            endp
p_int09         endp
                ;RS－232 通信口 OCH 中断处理程序
p_int0c         proc near
                sti
                push ax                 ;保护现场
                push bx
                push cx
                push dx
                push ds
                push cs
                pop ds                  ;DS=CS
                mov dx,3f8h             ;接收一字符
                in al,dx
                cmp al,ack              ;是 ACK?
                jz curpo                ;是,转去显示 ACK
                test flag,1             ;不是,已收到 ENQ?
                jnz rt1                 ;已收到 ENQ,转去比较本站地址
                cmp al,enq              ;未收到 ENQ,是 ENQ?
                jnz rtt                 ;不是,返回
                mov flag,1              ;是 ENQ,置标志
                jmp rtt
rt1:            cmp al,' c'             ;与本站地址比较
                jz curpo                ;是本站地址转去显示"请准备接收"!
                jmp rtt                 ;不是,返回
curpo:          mov ah,3                ;读取并保存现行光标位置
                mov bh,0
                int 10h
                push dx
                mov ah,2                ;置光标位置于底行
                mov dl,0
                mov dh,24
                mov bh,0
                int 10h
                test flag,1             ;判标志
                jnz p5                  ;有标志,转显示"请准备接收"
```

```
        mov bx,offset mseger        ;无标志,显示"连接成功"
        jmp p6
p5:     mov bx,offset mseg          ;显示"请准备接收"
p6:     mov cx,24
dp:     mov al,[bx]
        call dch
        inc bx
        loop dp
        mov al,7                    ;鸣叫
        call dch
        mov ah,2                    ;恢复光标位置
        pop dx
        mov bh,0
        int 10h
        mov flag,0                  ;标志清 0
rtt:    mov al,20h                  ;向 8259 送 EOI 控制字
        out 20h,al
        pop ds                      ;恢复现场
        pop dx
        pop cx
        pop bx
        pop ax
        iret                        ;中断返回
p_int0c endp
dch     proc near                   ;显示子程序
        push bx
        mov bx,0
        mov ah,14
        int 10h
        pop bx
        ret
dch     endp
        ;初始安装程序
start   proc far
        cli                         ;关中断
        push cs
        pop ds                      ;DS=CS
        mov ax,3509h                ;读原 INT09 向量
        int 21h
        lea di,bios_int09           ;填入 BIOS-INT09 单元
        mov [di],bx
        mov [di+2],es
        mov dx,offset p_int09       ;设置新的 INT09 向量
```

```
                mov ax,2509h
                int 21h
                mov dx,offset p_int1c          ;设置新的 INT1C 向量
                mov ax,251ch
                int 21h
                mov dx,offset p_int0c          ;设置新的 INT0C 向量
                mov ax,250ch
                int 21h
                sti                            ;开中断
                mov dx,offset start            ;保持驻留,返回 DOS
                mov ax,3101h
                int 21h
start           endp
code            ends
                end     start
```

例 5.3 用 BIOS 调用发送和接收字符

从键盘上接收一个字符,将其显示在屏幕上,并同时发向串口,之后又从串口读回并显示。若键盘接收的为扩展码,则显示"?",从串口读错也显示"?"。程序清单如下:

```
;       exam53.asm

                ;for PC_PC communication program
sseg            segment stack ' stack'
                db 128 dup(0)
sseg            ends
dseg            segment
textt           db ' PC_PC communication' ,' $'
com1            dw 0
cont            dw 16h                         ;发送字符个数 '
dseg            ends
cseg            segment
start           proc far
                assume cs:cseg,ss:sseg,ds:dseg
                push ds
                xor ax,ax
                push ax
                mov ax,dseg
                mov ds,ax
                mov ax,0600h                   ;全屏幕初始化为空白
                mov bh,07h                     ;属性值(黑底白字,正常显示)
                mov cx,0                       ;左上角坐标(0,0)
                mov dx,184fh                   ;右下角坐标(24,79)
                int 10h
                mov ah,2                       ;置光标位置
```

```
        mov dx,0b20h            ;行列坐标为(11,32)
        mov bh,0                ;0 页
        int 10h
        mov si,offset textt
next:   mov al,[si]             ;显示提示信息
        cmp al,' $'
        jz go_on
        mov bl,14h              ;显示属性(蓝底红字)
        call disp               ;调显示子程序
        inc si                  ;修改地址指针
        jmp short next
go_on:  mov ah,2                ;置光标位置
        mov dx,1400h            ;行列坐标为(20,0)
        mov bx,0                ;0 页
        int 10h
        mov ah,0
        mov dx,com1             ;通信口 1 初始化
        mov al,11000011b        ;波特率 4800,8 位数据,1 位停止位
                                ;无奇偶校验
        int 14h
send:   mov ah,0                ;接收键盘
        int 16h
        cmp al,0
        jnz zeor
        mov al,' ? '
zeor:   mov bl,26h              ;显示属性(绿底棕字)
        call disp               ;调显示子程序
        mov ah,1                ;发送字符
        mov dx,com1
        int 14h
        mov al,0                ;清除 AL 为 0
        mov ah,2                ;接收字符
        mov dx,com1
        int 14h
        cmp al,0
        jnz zero
        mov al,' ? '
zero:   mov bl,38h              ;显示属性(青底灰字)
        call disp               ;调显示子程序
        mov bl,07               ;置正常显示属性(黑底白字)
        mov al,20h              ;空格
        call disp
        dec cont
```

```
        jnz send
        ret
start   endp
disp    proc                    ;显示子程序
        push cx
        push ax
        mov ah,9                ;显示 AL 中的字符
        mov cx,1
        int 10h
        mov ah,0eh
        int 10h
        pop ax
        pop cx
        ret
disp    endp
cseg    ends
        end     start
```

5.1.5 串行通信接口实验

实验 5.1 以中断方式发送文件

实验要求：把例 5.1 所示的查询式通信程序改为中断式通信程序。

实验 5.2 查询式发送和接收字符

实验要求：

1. 把两台 PC 机通过 RS-232 接口插座按图 5.3 所示连接。

2. 用查询方式实现发送与接收。从键盘输入一个字符并发送，在对方的屏幕上显示出来。

3. 检查通信线状态寄存器，若接收数据有错，则清除掉接收到的数据，并显示一个“?”号。

5.2 异步串行通信接口扩展卡的设计

5.2.1 串行通信接口芯片 8251 的引脚与功能

8251 是 28 脚的双列直插式组件，它的管脚信号排列如图 5.6。整个芯片的信号可分为两组，一组与 CPU 有关，另一组与外设有关。

1. 与 CPU 有关的信号

$D_7 \sim D_0$：双向三态数据线。

$\overline{CS}$：片选信号。

C/$\overline{D}$：控制/数据选择。用于区分通过同一数据线传送的是命令、状态还是数据。此脚在高电平时，表示数据线上传送的是控制信号(命令或状态)，在低电平时，表示数据线上传送的是数据。

CLK：时钟输入。用于产生内部时序用的时钟，而不是用传送速率控制，在同步方式时，CLK 的频率至少要大于收发送器输入时钟频率的 30 倍，在异方式时，至少为收发送器输入频率的 4.5 倍，同时应大于 0.74MHz，小于 3.125MHz。

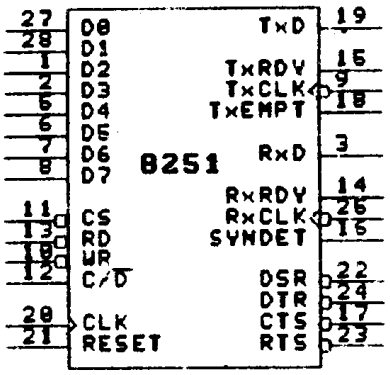


图 5.6　8251 芯片的引脚排列

Reset：复位信号。强迫 8251 进入空闲状态。

TXRDY：发送器准备好。它通知 CPU，8251 已准备接收待发送的数据字节。当采用中断方式进行数据传送时，TXRDY 状态线可作为中断请求信号使用；当采用查询方式进行数据传送时，CPU 可以使用状态读操作来检测状态寄存器中的 TXRDY 位状态。需要指出，实际上 TXRDY 引脚上的电平与状态寄存器中的 TXRDY 位是有区别的，当发送数据缓冲器空时，TXRDY 状态位置位，而 TXRDY 状态线当发送缓冲器空并允许 8251 发送（CTS＝0、TXE＝1）时才置位，在 CPU 发出数据写入 8251 后，TXRDY 就自动复位。

TXE：发送器空。用于表明在发送器中的并/串转换器已空，当 8251 从 CPU 接收数据时就自动复位，当字符发送完毕时变为高电平。

RXRDY：接收准备好。表明 8251 已从它的串行输入端接收了一个字符，并作好了传送 CPU 的准备，RXRDY 也可作为 CPU 的中断信号，或者 CPU 可用读状态操作来检测 RXRDY，在字符由 CPU 读入后，RXRDY 复位。

SYNDET：同步检则信号。它只用于同步工作方式，作为输入线用还是作为输出线用，是由程序规定的。在内同步方式下，它作输出线用，SYNDET 为高电平，表示 8251 已检测到所需的同步字符，因而同步已实现；在外同步方式下，它作输入线用，该线上的正跳变输入使 8251 在下一个 RXC 的上升沿开始装配字符。当芯片复位时，8251 被置成内同步方式，SYNDET 被置成低电平。

2. 与外设有关的信号

TXD：串行数据发送端。

$\overline{\text{TXC}}$：发送时钟输入端。用于控制字符发送速率。

RXD：串行数据接收端。

$\overline{\text{RXC}}$：接收时钟输入端。用于控制字符接收的速率，在异步方式，$\overline{\text{RXC}}$是 1，16，64 倍的波特率。

$\overline{\text{DTR}}$：数据终端准备好。是一个通用的输出信号，它表示 8251 已准备好。

DSR：数据设备准备好。是一个通用的输入信号，它表示外设或调制解调器已准备好。

$\overline{\text{RTS}}$：请求发送。是一个等效于 DTR 的输出信号，是串行接口准备给外设或调制解调器发送数据的信号。

$\overline{\text{CTS}}$：允许发送。是外设或调制解调器给 8251 的应答信号。

5.2.2　串行通信接口扩展实例

例 5.4　查询式串行通信接口扩展

如图 5.7 所示，串行通信接口以 8251 作为接口主芯片，再加上波特率发生器，RS-232C

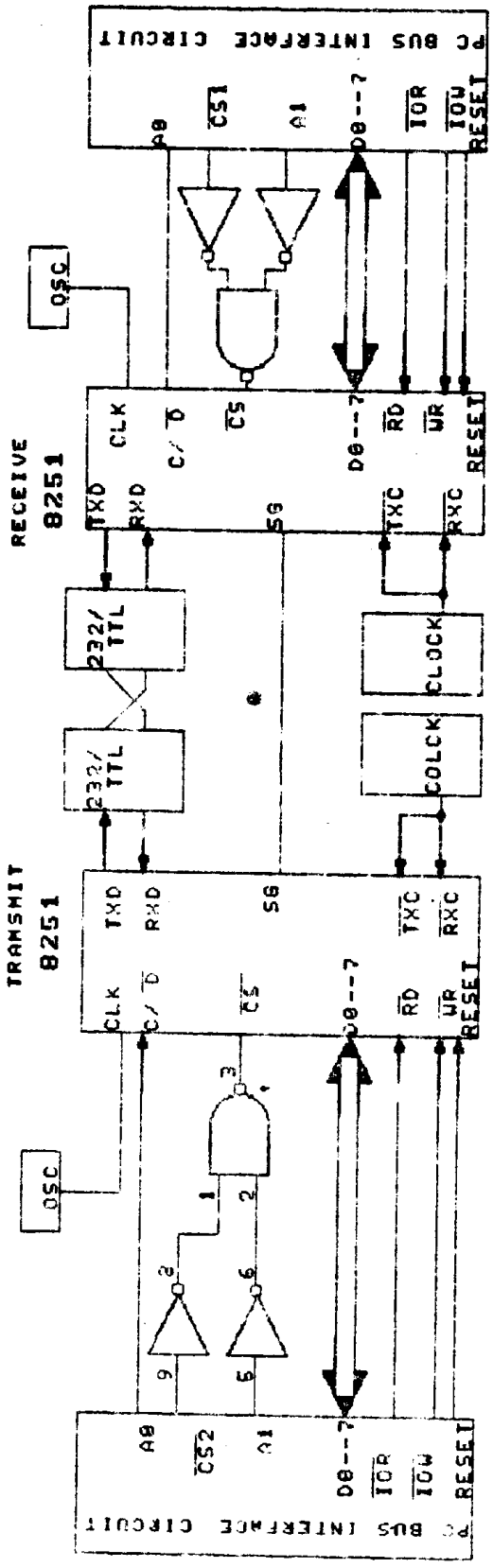


图 5.7 双机串行通信接口

与 TTL 电平转换电路，总线接口电路等就构成了一个串行通信接口卡。

波特率发生器可采用图 3.12 所示的电路，用晶振产生频率为 4.915MHz 的时钟，经 12 级 2 分频电路 CD4040 分频，最后可得频率为 1.2kHz 的时钟，正好为 75Hz×16，因此，最后的 8 个分频正好是 8 种速率的 16 倍，通过 DIP 开关选择构成 8 种可选速率的收发器。系统时钟 CLK 可用 2.4576MHz 的 2 分频输出。

RS-232C 与 TTL 电平的转换用芯片 1488 和 1489 实现，电路连接如图 5.8 所示。

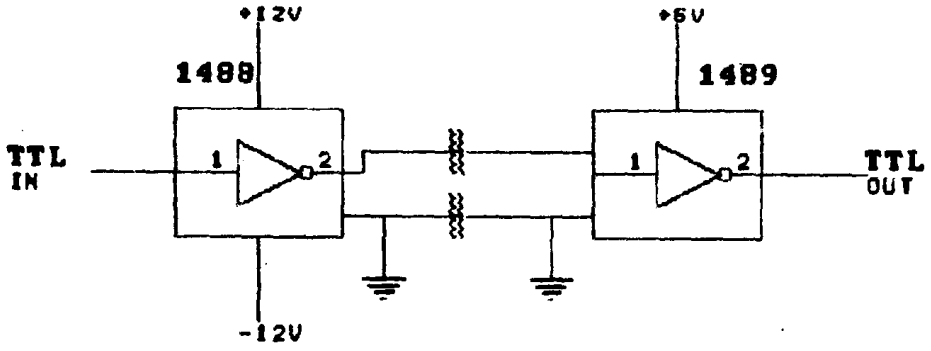


图 5.8 RS-232C 电平与 TTL 电平转换电路

两台 PC 机之间采用图 5.3 所示的三线连接方式。甲机发送，乙机接收，采用查询方式进行通信，把甲机 BUFFER 数据缓冲区的内容传送到乙机 BUFFER 数据缓冲区。接收程序和发送程序分开编写，每个程序段中包括 8251 初始化、状态查询和传送几部分。程序清单如下：

1. 发送程序：

```
;       exam54t.asm

data            segment
buffer          db '1','2','3','4','5','6','7','8','9','a'
blength         equ $-buffer
data            ends
stack           segment
                db 256 dup(?)
stack           ends
cseg            segment
                assume cs:cseg,ds:data,es:data,ss:stack
tra             proc far
start:          push ds
                xor ax,ax
                push ax
                mov ax,data
                mov ds,ax
                mov es,ax
                mov ax,stack
```

```
                mov ss,cx
                lea si,buffer
                mov cx,blength
                mov dx,269h             ;控制口
                mov al,00h              ;空操作
                out dx,al
                mov al,40h              ;复位
                out dx,al
                nop
                mov al,0deh             ;方式字,8 位数据位,1 位起始位,2 位停
                                        ;止位
                out dx,al               ;奇校验,波特率系数为 16
                mov al,37h              ;命令字 RTS,ER,RXE,PTR,TXEN 均置位
                out dx,al
l1:             mov dx,269h             ;状态口
                in al,dx                ;查状态位 D0(TXRDY)=1
                and al,01h
                jz l1                   ;发送未准备好,则等待
                mov dx,268h             ;数据口
                mov al,[si]             ;发送准备好,则从发送区取一字节发送
                out dx,al
                call disp               ;显示发送数据
                inc si                  ;修改内存地址
                dec cx                  ;字节数减一
                jnz l1                  ;未发送完,继续
                ret                     ;已送完,返回 DOS
tra             endp
disp            proc near
                push ax
                push bx
                push cx
                mov ah,09
                mov cx,1
                mov bh,0
                mov bl,28
                int 10h
                pop cx
                pop bx
                pop ax
                ret
disp            endp
cseg            ends
                end  start
```

2. 接收程序：

```
;       exam54r.asm

data            segment
buffer          db 10 dup(?)
blength         equ $ -buffer
data            ends
stack           segment para stack ' stack'
                db 256 dup(?)
stack           ends
cseg            segment
                assume cs:cseg,ds:data,es:data,ss:stack
rec             proc far
begin:          push ds
                xor ax,ax
                push ax
                mov ax,data
                mov ds,ax
                mov es,ax
                mov ax,stack
                mov ss,ax
                lea di,buffer
                mov cx,blength
                mov dx,265h                 ;控制口
                mov al,00h                  ;空操作
                out dx,al
                mov al,40h                  ;复位
                out dx,al
                nop
                mov al,0deh                 ;方式字
                out dx,al
                mov al,14h                  ;命令字,ER,RXE置位
                out dx,al
12:             mov dx,265h                 ;状态口
                in al,dx                    ;查状态位 D2(RXRDY)=1
                and al,02h
                jz 12                       ;接收未准备好,则等待
                mov dx,264h                 ;数据口
                in al,dx                    ;接收准备好,则接收一字节
                mov [di],al                 ;并存入接收区
                call disp                   ;显示接收数据
                inc di                      ;修改内存地址
                loop 12                     ;未接收完,继续
```

```
                ret                             ;已接收完,返回 DOS
rec             endp
disp            proc near
                push ax
                push bx
                push cx
                mov ah,09
                mov cx,1
                mov bh,0
                mov bl,28
                int 10h
                pop cx
                pop bx
                pop ax
                ret
disp            endp
cseg            ends
                end begin
```

5.2.3 串行通信接口扩展实验

实验 5.3 8251 串行通信

一、实验原理

8251 的接口电路如图 5.9 所示。8251 的 CLK 时钟信号采用 1.8432MHz 的标准频率，并经 64 分频(可调)后作为 8251 的接收时钟，分频器选用 74LS393 芯片。C/$\overline{D}$ 端接地址线 A_0，$\overline{CS}$信号由 A_3～A_7 共同产生。8251 的 TXRDY(发送器准备好)和 RXDY(接收器准备好)产生中断请求信号，它们先或后接到系统总线上的 NMIR 或 INTRQ 端。8251 的 SYNDET 信号线外接，可以使 8251 工作在同步通信方式。8251 通信线与 RS-232 接口之间有电平转换器 MC1488 和 MC1489，与 RS-422 接口之间有电平转换器 MC3487 和 MC3486。

二、实验要求

1. 根据电路原理框图设计线路图

2. 编写 8251 的数据发送和接收程序，PC 机采用查询方式通信，8251 采用中断方式通信，通信双方传送十个数据，字符长度为 8 位、2 个停止位、奇校验，波特率为 1800。

3. 运行 8251 的数据发送程序，并在输出端 TXD 上用示波器观察输出波形。

4. 用信号线把模板上 8251 的 RC-232 接口与 PC 机上的 RS-232 端口相接。一方发射，一方接收，并观察接收数据是否正确。

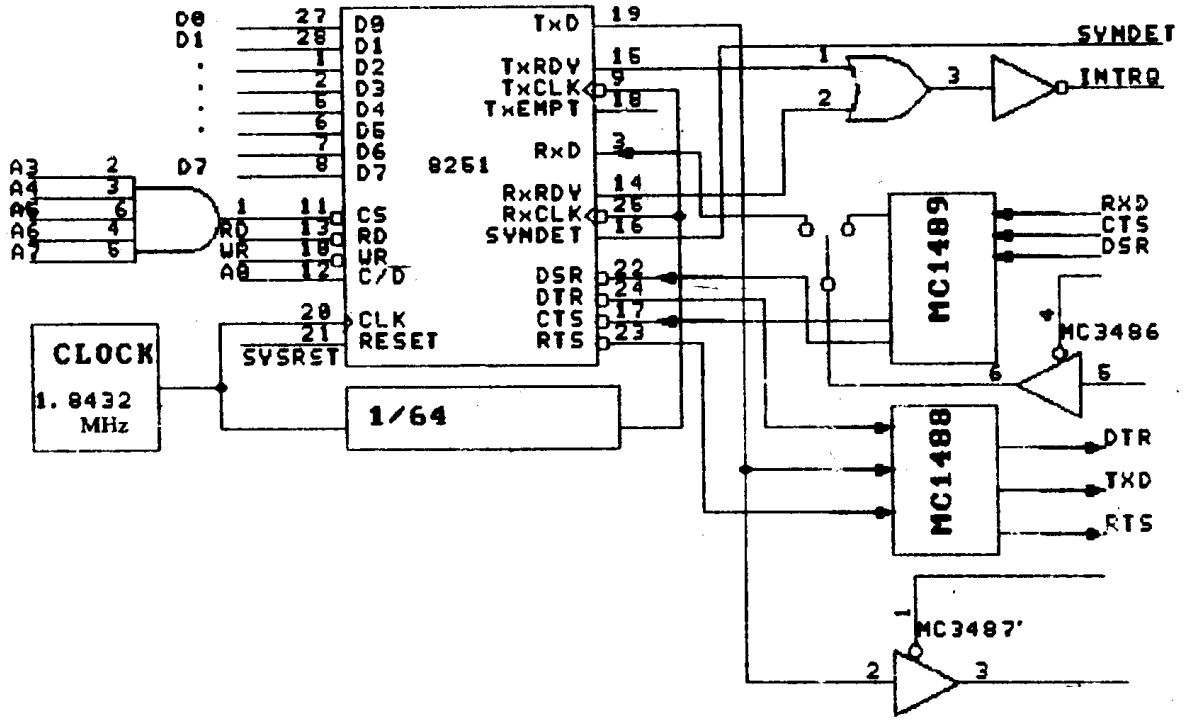


图 5.9　8251 串行通信接口实验原理框图

5.3　并行通信接口扩展卡的设计

5.3.1　并行接口芯片 8255 的引脚与功能

8255 是 40 脚的双列直插式组件，它的管脚信号排列如图 5.10。

D_7～D_0：双向数据总线，用于与 CPU 进行数据传送。

$\overline{CS}$：片选，当为低电平时，允许 8255 和 CPU 之间进行数据传送。

$\overline{RD}$：读控制，当为低电平时，8255 通过数据总线向 CPU 发送数据或状态。

$\overline{WR}$：写控制，当为低电平时，8255 通过数据总线写入由 CPU 送出的数据或控制字。

A_0，A_1：端口地址，用于对三个端口地址和控制寄存器寻址。

Reset：复位，使所有内部寄存器清零。

PA_1～PA_0：端口 A，与外设接口的一组 I/O 并行口。

PB_1～PB_0：端口 B，与外设接口的另一组 I/O 并行口。

PC_7～PC_0：端口 C，与外设接口的另一组 I/O 并行口。

V_{CC}：电源（+5V）。

GND：地。

5.3.2　并行通信接口扩展实例

例 5.5　查询式并行通信接口扩展

接口电路的连接如图 5.11 所示，用 8255 为主芯片构成并行通信接口。甲机发送，乙机接收，采用查询方式通信。甲机一侧的 8255A 采用 1 方式工作，端口地址为 264～267H，乙机一侧的 8255A 采用 0 方式，端口地址为 26C～26FH。甲机 C 口中分配作联络的专用应答线 PC_7 ($\overline{OBF}$)与乙机的 PC_4 相连，这样乙机就通过查询 PC_4 是否为 0，若为零表示甲机已把数据发出，乙机可以接收数据，否则等待。甲机的 PC_6($\overline{ACK}$)与乙机的 PC_0 相连，这样乙机就可通过使 PC_0 置零，产生$\overline{ACK}$信号发给甲机，$\overline{ACK}$由高电平变低电平(下降沿)将$\overline{OBF}$置高电平，使$\overline{OBF}$无效，表示甲机数据没有准备好。乙机再把 PC_0 置 1，使$\overline{ACK}$由低电平变高电平(上升沿)使 INTRA 变高电平，产生中断请求，甲机查询到 INTRA 变高则通过 A 口向乙机发送数据，发送数据的 OUT 指令产生的$\overline{WR}$信号又会使$\overline{OBF}$变低，通知乙机取走数据。通信程序清单如下：

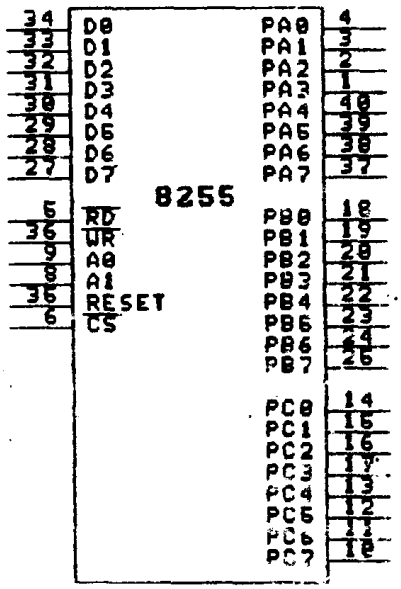


图 5.10　8255 芯片的引脚排列

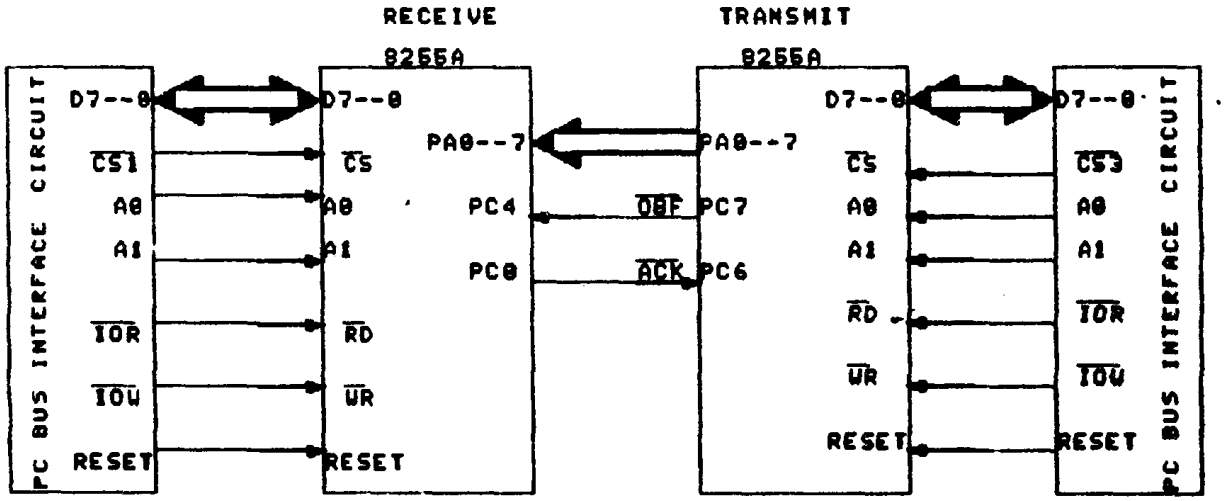


图 5.11　双机并行通信接口

1. 甲机发送程序

```
;       exam55t.asm

data        segment
buffer      db ' 1' ,' 2' ,' 3' ,' 4' ,' 5' ,' 6' ,' 7' ,' 8' ,' 9' ,' a'
blength     equ $ -buffer
data        ends
stack       segment para stack ' stack'
            db 256 dup(?)
stack       ends
code        segment
```

```
        assume cs:code,ds:data,es:data,ss:stack
main    proc far
begin:  push ds
        xor ax,ax
        push ax
        mov ax,data
        mov ds,ax
        mov es,ax
        mov ax,stack
        mov ss,ax
        lea bx,buffer
        mov cx,blength
        mov dx,267h                 ;8255A 命令口
        mov al,10100000b            ;工作方式字
        out dx,al
        mov al,0dh                  ;置中断允许 INTEA=1(PC6=1)
        out dx,al
1:      mov dx,266h                 ;8255A 状态口
        in al,dx                    ;查 INTRA=1? (PC3=1?)
        and al,08h
        jz 1                        ;若未准备好发送数据,则等待
                                    ;准备好,则向 A 口写数
        mov dx,264h                 ;8255A 口地址
        mov al,es:[bx]              ;从内存取数
        out dx,al                   ;通过 A 口向乙机发送数据
        call disp                   ;显示发送数据
        inc bx                      ;内存地址加一
        dec cx                      ;字节数减一
        jnz 1                       ;字节未完,继续
        ret
main    endp
disp    proc near
        push ax
        push bx
        push cx
        mov ah,09
        mov cx,1
        mov bh,0
        mov bl,28
        int 10h
        pop cx
        pop bx
        pop ax
```

```
        ret
disp    endp
code    ends
        end   begin
```

2. 乙机接收程序

```
;     exam55r.asm

data        segment
buffer      db 10 dup (0)
blength     equ $-buffer
data        ends
stack       segment para stack ' stack'
            db 256 dup(?)
stack       ends
code        segment
assume      cs:code,ds:data,es:data,ss:stack
main        proc far
begin:      push ds
            xor ax,ax
            push ax
            mov ax,data
            mov ds,ax
            mov es,ax
            mov ax,stack
            mov ss,ax
            lea bx,buffer
            mov cx,blength
            mov dx,26fh                 ;8255A 命令口
            mov al,10011000b            ;工作方式字
            out dx,al
            mov al,00000001b            ;置 ACK=0(PC0=1),因为尚未收
                                        ;到数据
            out dx,al
11:         mov cx,26eh                 ;8255A 状态口
            in al,dx                    ;查 OBF=1? (PC4=0)
            and al,10h                  ;即查甲机是否有数据发来
            jnz 11                      ;若无数据发来,则等待
                                        ;有数据,则从 A 口读数
            mov dx,26ch                 ;8255A 口地址
            in al,dx                    ;从 A 口读入数据
            mov es:[bx],al              ;存入内存
            mov dx,26fh                 ;产生 ACK 信号并发回给甲机
            call disp                   ;显示接收数据
```

```
                mov al,0000000b          ;PC0 置 0
                out dx,al
                nop
                nop
                mov al,00000001b         ;PC0 置 1
                out dx,al
                inc bx                   ;内存地址加 1
                dec cx                   ;字节数－1
                jnz l1                   ;字节未完,则继续
                ret
main            endp
disp            proc near
                push ax
                push bx
                push cx
                mov ah,09
                mov cx,1
                mov bh,0
                mov bl,28
                int 10h
                pop cx
                pop bx
                pop ax
                ret
disp            endp
code            ends
                end begin
```

5.3.3 并行通信接口扩展实验

实验 5.4 中断方式通信接口扩展

实验要求：

1. 改进图 5.11 所示的并行通信接口电路以实现中断方式通信接口的需要。
2. 编制中断方式通信接口软件。

实验 5.5 8255 与键盘和显示器接口

一、实验原理

实验要求：

实验电路图如图 3.23 所示，8255 的 PA 口用于读取键值，PB 口用于输出显示段码，PC 口只用低 3 位作显示与键盘和扫描译码，最高位用于显示消隐（“0”＝消隐），因此 8255 的控制字应为 90H。

二、实验要求

1. 根据图 3.23 确定 8255 的端口地址。

2. 编 8255 初始化程序，8255 驱动的键盘和显示程序。键盘程序应包括如下的三个部分：判键、识键和键处理。

5.4 PC 系列微机与多台 MCS-51 单片机间的通信

随着计算机技术的迅猛发展和计算机应用的广泛深入，分布式系统正成为各种大、中型信息处理，实时监控系统的主要形式。在这些分布式系统中有一类很典型的系统，这类系统一般由两级计算机组成，即上位机和前端机。上位机可以是 PC 及其兼容机，其人机界面好，软件资源丰富。主要用于过程监控、优化计算和数据管理等。前端机可以是单片机，其价格低廉、系统构成灵活、抗干扰能力强，主要用于数据采集、数据处理和实时控制等。这类系统的性能价格比高，充分利用单片机和微机的各自优点。上位机还可以通过微机局域网、PABX 或广域网与其他计算机相连，共享资源，构成规模更大、功能更强的分布或应用系统。

本书就是充分利用这两类计算机各自所具有的特点，以一个较典型的计算机多机通讯系统作为实例，来研究多机系统的实现方法，这一系统的特点是：利用 PC 机的 RS-232C 异步通讯口直接与多台 MCS-51 单片机组成的前沿控制机进行联网，主机与前沿机之间不需要其他中间适配器，系统接口简单，主要技术问题均通过软件方法得以解决。

5.4.1 多机通信原理

一、8031 实现多机通信的原理

1. 8031 的串行口控制寄存器 SCON

MCS-51 系列单片机有一个全双工的串行口，它不仅具有双机之间的串行通信能力，而且还可以实现多机通信。由 MCS-51 构成的多机系统常采用主从式结构。主机与各从机之间可实现全双工通信，而各从机之间只能通过主机交换信息。在多机通信中，要保证主机与所选择的从机实现正确可靠的通信，必须保证通信接口具有识别功能。MCS-51 的串行口控制寄存器 SCON 中的多机通信控制位 SM2 就具有这一功能。

SCON 的格式如下：

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

• SM0、SM1 为串行口工作方式选择位，有四种方式可供选择。

• SM2 为方式 2，3 的多机通信控制位。若 SM2＝1 可以进行多机通信，这时出现两种情况：收到的第 9 位数据 RB8＝1，数据装入接收缓冲器 SBUF，内部硬件产生接收中断请求(RI 置位)；收到的第 9 位数据 RB8＝0，则不产生接收中断请求(不置位 RI)，信息被抛弃。若 SM2＝0，则不论 RB8 是 0 还是 1 都将产生中断请求(RI 置位)。这样就可以区分所接收的信息是地址还是数据，据此可实现多机通信。

• TB8 为方式 2，3 中要发送的第 9 位数据。是可编程位，即可通过程序改变它的状态。在多机通信中用来区别发送的数据是地址帧还是数据帧。

• RB8 为方式 2，3 中所接收的第 9 位数据。在多机通信中用来作地址/数据标识位。

2. 8031 串行口数据格式

8031 的串行口在工作方式 3 下，波特率是可变的，通信数据长度为 9 位，每发送或接收一帧信息 11 位(1 位起始位、8 位数据位、1 位附加校验位、1 位停止位)。其中附加的第 9 位

数据，在发送端由串行控制寄存器 SCON 中的 TB8 产生，在接收端自动将该位数据传送到 SCON 中的 RB8 中。该位数据可以由程序设定为‘0’或‘1’来实现多机通信中的数据帧和地址帧的分离。我们设定‘1’为地址帧标识，‘0’为数据帧标识。

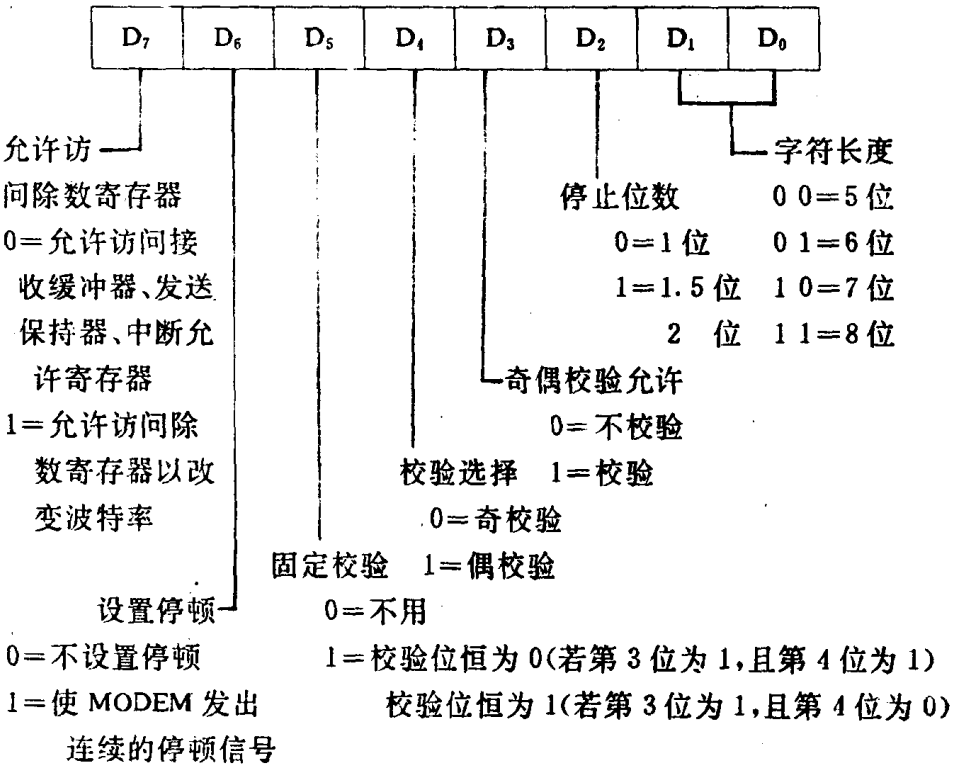
二、PC 机实现多机通信的原理

IBM-PC 等各种兼容计算机中的异步通信口，均是由 INS8250 作为通信的主控部件，它的内部是由多个可访问的寄存器进行管理。由于在 8031 串行口工作方式 3 中的数据传输格式是由厂家设计好的，一帧数据 11 位，在与其他计算机进行异步通信活动其灵活性受到了很大的限制，只有符合同种数据格式的计算机才能与之发生通信活动。为了避免不必要的系统硬件开销，使 PC 机能与 8031 不通过中间控制器而直接用异步通信口进行多机通信，必须满足两个条件：

其一：PC 机必须具有与 8031 同样的数据格式；

其二：第 9 位数据必须是可编程控制的，以区别是地址帧和数据帧。

这两个条件缺一不可。可通过对 INS8250 的线路控制寄存器 3FB 进行编程来实现与 8031 同样的通信格式。8250 内部的线路控制寄存器 3FB 的格式如下：



由上可知，通过对线路控制寄存器的写入，可以规定异步通信口的数据格式。可见，INS8250 在编程使用中比 8031 具有更大的灵活性，使用范围更大。

要实现 PC 机与 8031 直接多机通信，其关键就在于控制它的线路状态，以保持与 8031 串行口工作方式相一致的数据格式和控制校验位的值(第 9 位数据)，根据 8250 线路控制寄存器的结构特点，在编程中作如下设定：

1. 设定：D_1，D_0 为 1，1，用来控制数据字长 8 位。

2. 设定：D_2 为 0，产生 1 位停止位。

3. 设定：D_5，D_4，D_3 为 1，0，1 控制校验位为“1”，即第 9 位数据为 1，作为地址帧的发送和

接收标志。

4. 设定：D_5，D_4，D_3 为 1，1，1，控制校验位为“0”，即第 9 位数据为 0，作为数据帧的发送和接收标志。

其中 3，4 两点设置特别重要，它是能否实现可控第 9 位数据的关键，有了可控的第 9 位，实现 PC 机与 MCS-51 直接多机通信就不复杂了。

例如：发送 0A 的地址帧，由下列程序实现：

```
MOV DX,3FBH          ;线路控制寄存器
MOV AL,2BH           ;设置校验位为 1,8 位字长
OUT DX,AL            ;1 位停止位
MOV DX,3F8H          ;指向发送缓冲器
MOV AL,0AH           ;0AH 为地址号
OUT DX,AL            ;启动发送
```

发送数据帧，由下列程序实现：

```
MOV DX,3FBH          ;指向线路控制寄存器
MOV AL,3BH           ;设置校验位为 0,8 位字长
OUT DX,AL            ;1 位停止位
...
```

5.4.2 波特率的设置

在一个通信系统中，发送端和接收端的波特率必须设置相同，才能保证可靠的通信。

一、8031 波特率的设置

8031 的串行口工作在方式 3 时，波特率的设置由定时器 1 置方式 2 自动重新装入。波特率的运算公式为：

$$\text{波特率}=\frac{2^{SMOD}}{32}\times\frac{F_{osc}}{12\times(2^N-TH_1)}$$

其中：F_{osc}——时钟振荡频率

TH_1——定时器 T_1 的预置数

N——所用计数器的位数

SMOD——波特率加倍

二、PC 机异步通信口波特率的设置

设置波特率就是将 1.8432MHz 的时钟输入频率采用分频的办法来得到要求的波特率，分频所用的除数是由 CPU 分两次写入除数锁存器的高位(MSB)和低位(LSB)部分，除数可用下面的公式求得：

$$\text{除数}=1843200/(\text{波特率}\times 16)$$

根据以上两个公式计算出常用的波特率，PC 机除数值及 8031 定时器 1 计数常数(晶振频率为 6MHz)如表 5-2，并给出每种波特率下两种机型之间通信的误差，一般允许最大的波特率误差不能大于 4.5%。从表中可看出，波特率小于 2000Bit/s 误差较小，实验也证明了在 2000Bit/s 时通信正常，大于 2000Bit/s 时误差明显加大，通信不能成功，与理论分析一致。产生误差的主要原因是由于 8031 的晶振频率不是这些常用波特率的整数倍，分频后有误差。如采用特殊频率的晶振如 11.059MHz，则可降低误差，提高通信速率。

表 5-2 波特率误差

波特率	PC 除数值	8031 TH1	波特率误差
110	04 17	72	0.00%
150	03 00	99	1.13%
300	01 80	CC	0.16%
600	00 C0	E6	0.16%
1200	00 60	F3	0.16%
1800	00 40	F7	3.55%
2000	00 3A	F8	2.34%
2400	00 30	F9	6.09%
3600	00 20	FC	8.51%
4800	00 18	FD	8.51%
7200	00 10	FE	8.51%
9600	00 0C	—	—

5.4.3 通信协议

在通信系统的设计时，为了可靠地使主机与从机间发生有效的通信，需要对它制定一适用于系统的约定保证。我们将其定义为通信协议：

(1)PC 机在这一系统中是作为指挥、调度者、对各从机进行访问，发布各种命令和接收各种从机发来的各种信息。

(2)分别给各单片机安排一地址号，如 01,02,…,n。

(3)所有单片机的 SMZ 位置 1，处于接收地址帧状态。

(4)PC 机首先对访问的从机进行“呼号”，呼号时发一帧地址帧(数据第 9 位为 1)。

(5)各单片机产生接收中断，在中断服务子程序中各自将所接收的地址与本机的地址相比较，对于地址相符的单片机，使 SMZ 位清“0”以接收 PC 机随后发来的命令信息，同时将本机的地址发回给 PC 机，即表示本机与主机“接通”，呼号成功。对于地址不相符的单片机仍保持 SMZ＝1 的状态，对主机随后发来的命令信息不予理睬，直至发送新的地址帧。

(6)PC 机在接到从机发回的地址后，即呼号成功，主机便发送命令帧或数据帧(TB8＝0)。命令有两种类型，一种是“发送数据”命令，另一种是“接收数据”命令，对于“发送数据”，则主机在“呼号”成功后，发送一帧或两帧数据长度(视数据量而定)，从机接到该长度后，将其作为长度指针，然后再发送真正的数据，而对于“接收数据”命令，则主机发出命令后，便等待数据长度和数据。

(7)在数据的传输过程中，由于“校验位”在此作为“地址帧”和“数据帧”辨识位，使系统失去了辨错能力，为了解决这一问题，在发送端和接收端各设置了一个单元作为“检查和”，在发送和接收前将该单元清“0”，然后每发送或接收一帧数据，便将其加到“检查和”单元中，相加时，不必考虑标志的情况，如果通信正常，则在发送和接收两端的“检查和”单元的值应该是一致的。在数据传送完毕后，两边通过校对“检查和”单元来判断通信成功与否，若通信

失败，则重新发送，直到正确为止。

5.4.4 多机通信系统实例

例 5.5 PC 机与 MCS-51 组成的多机通信系统

一、系统硬件设计

图 5.12 所示的是一个典型的 PC 机与 MCS-51 构成的多机接口的系统。它由一台 PC 机

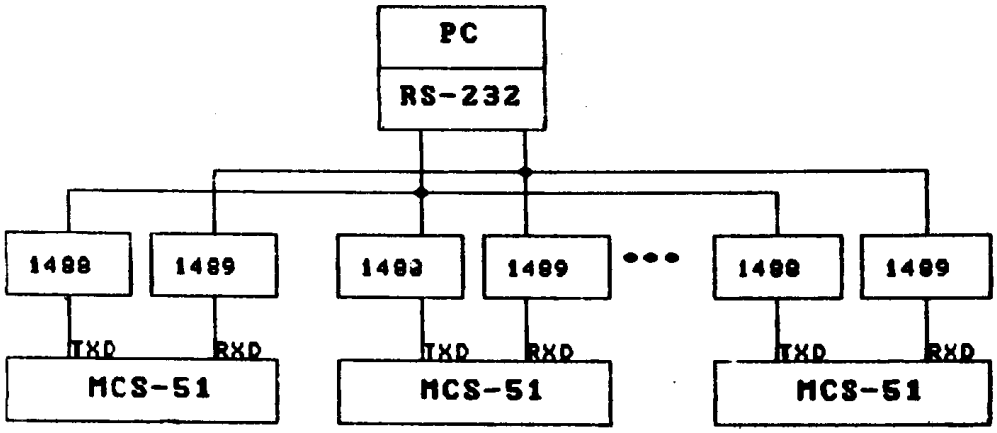


图 5.12 多机通信系统

或兼容机与多台 MCS-51 单片机组成，每个 MCS-51 单片机为一个独立单元，并有 A/D、D/A 等 I/O 功能，有独立监控，用于对控制对象进行控制和采集。本系统主机与前沿机间的联接距离约 15 米左右，可直接应用 RS-232C 通讯口进行传送，但由于 8031 的串行口是标准 TTL 电平输出，为使其与 RS-232C 电平接口，在 8031 串行口联有 1488 和 1489 转换成 RC-232C 电平，以实现电平匹配。

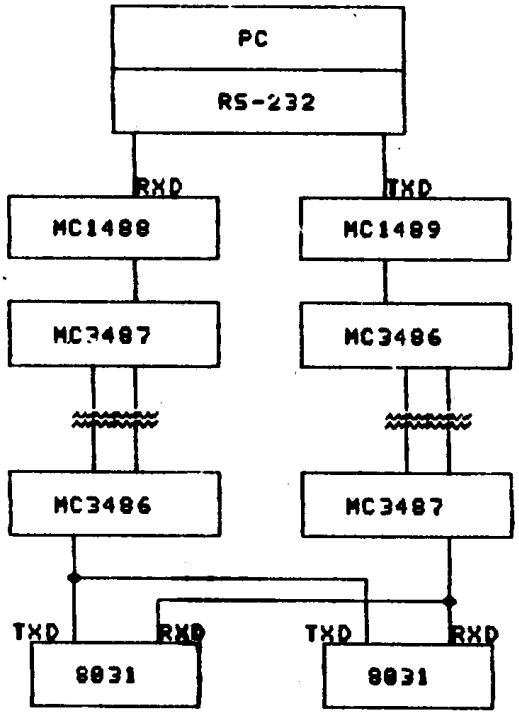


图 5.13 RS-232 电平与 RS-422 电平转换

图 5.12 的结构只是原理型的应用，由于 IBM-PC 的异步通信标准，理论上规定在“0”调制解调状态下，传送距离只能是 15 米左右。而在传输过程中，特别是工业过程控制现场难免存在各种干扰源，超过这个距离，传输的误码率就会升高，使系统工作不可靠，因此，在分布式系统中应用这种标准是不切实际的，对于通讯距离适中的系统，有效的方法是把 RS-232C 总线转换成 RS-422 总线，以提高系统的传输速度和传输距离，转换电路见图 5.13，它能以 1Mbps 的波特率传送数据，通信距离达 1km 以上，适应于一般工业过程控制的需要。

二、系统软件设计

1. 主机通信软件设计

主机软件可由高级语言和汇编语言设计，一般系统管理程序，如界面设计、数据处理等模块由高级语言编写，而系统通信软件由汇编语言编写。

在本应用实例中，通信管理程序由 BASIC 语言编写，通信程序由汇编语言编写，程序框图如图 5.14 所示。主程序将通信所使用的各种参数传送给汇编语言的参数区，由主程序调用汇编通信程序，将随机数据文件"DATA"中的一个记录中的串变量 A$ 作为数据发送给单片机，并从单片机接收一个字符串给一个记录的 B$ 变量。程序清单如下：

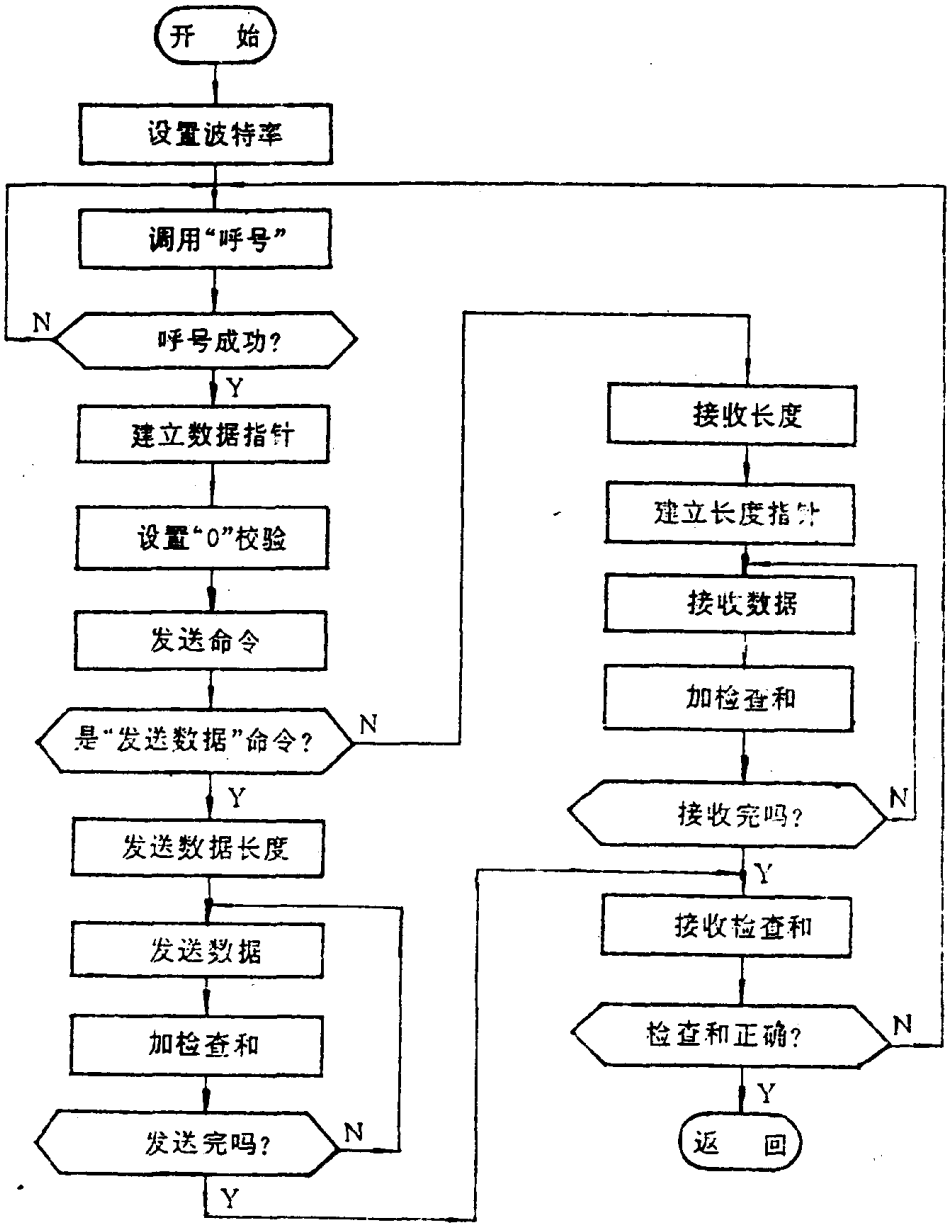


图 5.14 主机通讯程序框图

```
5     REM     exam56.bas
```

```
10   REM * * * * * * 通信管理程序 * * * * * *
20   DEF SEG=&H9FA0: BLOAD "exam56.bas", &H0: HB=&H0
30   BUFF0=&H3: BUFF1=&H4: BUFF2=&H5: BUFF4=&H7
40   CLS: INPUT "有数据发送吗?" , Hs
50   IF Hs= "Y" or "y" THEN 60 ELSE 180
60   POKE BUFF0, &HFF
70   INPUT "数据发送给哪号分机", FJH
80   OPEN "data" AS#1 LEN=400
90   FIELD #1, 200 AS As, 200 AS Bs
100  INPUT "请输入记录号",CODE%
110  GET #1, CODE%: CLOSE
120  PRINT As:V=LEN(As)
130  POKE BUFF2, FJH:POKE BUFF1,V
140  FOR I=1 TO V
150  Vs=MIDs(As,I,1):P=ASC(Vs)
160  POKE BUFF4,P:BUFF4=BUFF4+1:NEXT I
170  CALL HB:GOTO 40
180  POKE BUFF0,&HC
190  CLS:INPUT "接收哪号分机发来的数据?", FJH
200  POKE BUFF2,FJH
210  OPEN "data" AS#1 LEN=400
220  FIELD #1, 200 AS A$, 200 AS B$
230  INPUT "请输入记录号" CODE%
240  CALL HB:.V=PEEK(BUFF1):FOR I=1 to BUFF1
250  V2s=CHR$(PEEK(BUFF4)): BUFF1=BUFF1-1: BUFF4=BUFF4+1
260  V3s=V3s+V2s: NEXT I
270  LSET Bs=V3s: PUT #1, CODE%: CLOSE
280  GOTO 40

;    exam57.asm

cseg      segment para' code'
f25       proc far
          assume cs:cseg,ds:cseg,es:cseg,ss:cseg
          org 0000h
          jmp start
buff0     db ?                          ;命令字
buff1     db ?                          ;数据长度
buff2     db ?                          ;叫号单元
buff3     db ?                          ;检查和单元
buff4     db 200 dup (?)                ;数据区
buff5     db ?                          ;叫号成功
```

```
start:  mov dx, 3fbh
        mov al,80h                      ;设 DLAB=1
        out dx, al
        mov dx, 3f8h                    ;波特率 2400
        mov al, 30h
        out dx, al
        mov dx,3f9h
        mov al, 00h
        out dx, al
bb14:   call jhsub                      ;呼号子程序
        mov al, buff5
        cmp al,00h
        jnz bb14
        mov ah,1
        mov ah, buff0
        mov dx, 0
        int 14h                         ;发送命令字
bb10:   mov buff3, 0
        mov ch, 0
        mov cl, buff1
        mov si, offset buff4
        mov al, buff0
        test al, 0ffh
        jz bb372                        ;接收命令转
        mov dx, 3fbh                    ;发送命令
        mov al, 3bh
        out dx, al                      ;设置"0"校验
        mov ah, 1
        mov al,cl
        mov dx, 0
        int 14h                         ;发送长度
bb11:   mov ah, 1h
        mov al, [si]
        mov dx, 0
        int 14h                         ;发送数据
        add buff3, al
        inc si
        loop bb11
bb12:   mov ah, 3h
        mov dx, 0
        int 14h                         ;检测状态
        and ah, 1
        jz bb12
```

```
        mov ah, 2
        mov dx, 0
        int 14h                 ;接收检查和
        sub al, buff3
        jnz bb14
        jmp bb15
bb372:  call qs                 ;接收长度
        mov buff1, al
        mov cl, al
bb373:  call qs                 ;接收数据
        mov [si], al
        inc si
        loop bb373
        call qs                 ;接收检查和
        sub al, buff3
        jnz bb14
bb15:   ret
f25     endp
jhsub   proc  near              ;叫号子程序
        mov buff5, 0fh
        mov dx, 3fbh
        mov al, 2bh
        out dx, al              ;设置"1"校验值
        mov ah, 1
        mov al, buff2
        mov dx, 0
        int 14h
jhsub1: mov ah, 3
        mov dx, 0
        int 14h
        and ah, 1
        jz jhsub1
        mov ah, 2
        mov dx, 0
        int 14h
        sub al, buff2
        jz jhsub3
        jmp jhsub4
jhsub3: mov buff5, 0            ;呼号成功
jhsub4: ret
jhsub   endp
qs      proc near               ;接收数据子程序
        mov ah, 3
```

```
        mov dx, 0
        int 14h
        and ah, 1
        jz qs
        mov ah, 2
        mov dx, 0
        int 14h
        add buff3, al
        ret
qs      endp
cseg    ends
        end     start
```

2. 单片机的通信软件设计

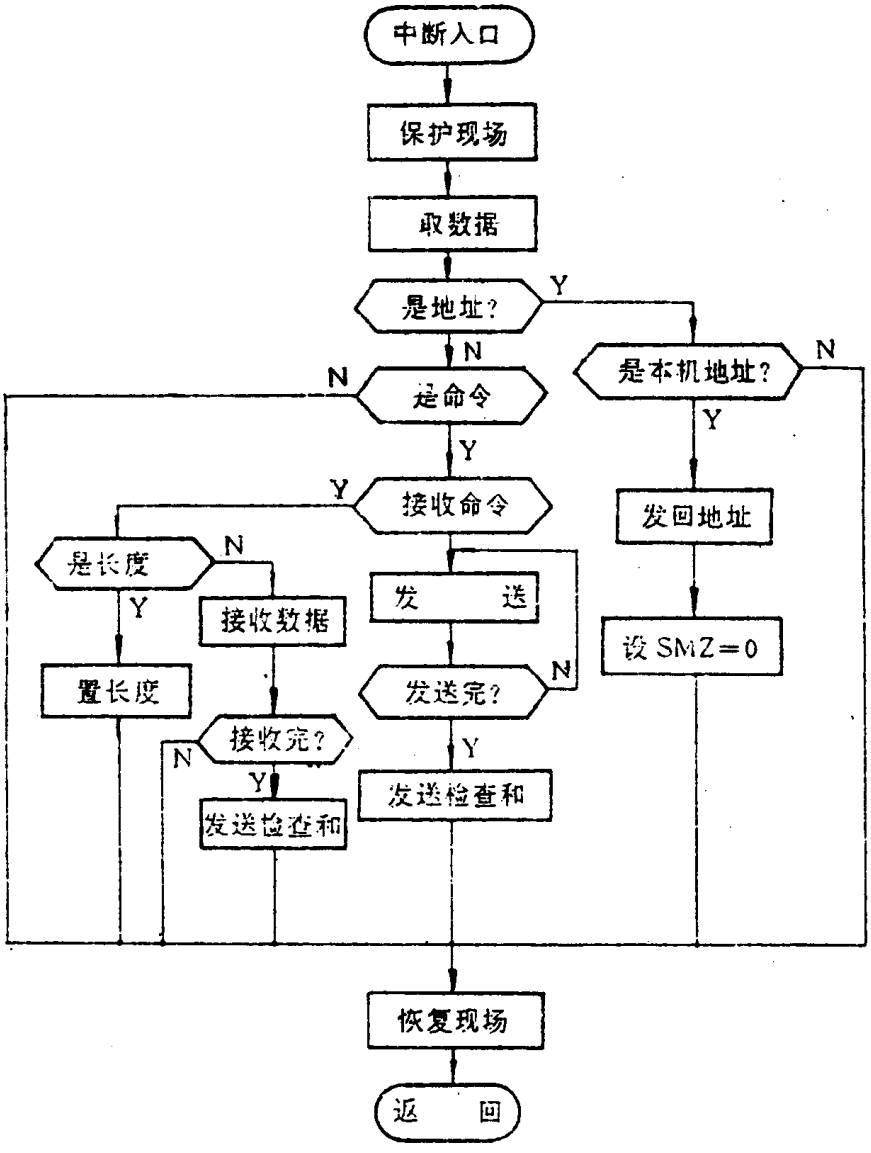


图 5.15 前沿机中断服务程序

单片机部分的通信程序采用可变波特率的串行口工作方式 3，定时器 T_1 为可自动装入参数的工作方式 2，采用串行口中断方式，波特率设置与 PC 机匹配，图 5.15 是该中断服务程序的框图，并给出 8031 源程序清单。

```
;       exam58.asm

                org 0000h
                sjmp bgin
                org 0023h
                sjmp sio
                org 0030h
bgin:           mov tmod, #20h
                mov th1, #0f9h
                mov tl1, #0f9h
                mov scon, #0f0h
                mov pcon, #00h
                setb tr1
                setb ea
                setb es
                setb 00h
                mov a, p1
                anl a, #0f0h
                mov 30h, a
wait:           sjmp wait
sio:            clr ri
                push psw
                push acc
                mov psw, #08h
                jnb 00h, sio2
sio1:           mov a, sbuf
                xrl a, 30h
                jnz back
                mov a, 30h
                mov sbuf, a
                clr 00h
                clr sm2
wat1:           jnb ti, wat1
                clr ti
                sjmp back
sio2:           jb 01h, sio3
                setb 01h
                mov a,sbuf
                mov 31h, #00h
                clr c
```

```
                subb a, #0ffh
                jz sio5
                sjmp back
sio3:           jb 02h, sio4
                setb 02h
                mov a, sbuf
                mov 32h, a
                mov r1, a
                mov dptr, #0f00h
                simp back
sio4:           mov a, sbuf
                mov a, sbuf
                movx @dptr, a
                inc dptr
                add a, 31h
                mov 31h, a
                djnz r1, back
                mov a, 31h
                mov sbuf, a
wat2:           jnb ti, wat2
                clr ti
                sjmp back
sio5:           mov dptr, #0f00h
                mov r1, 32h
                mov a, r1
                mov sbuf, a
wat3:           jnb ti, wat3
                clr ti
sio6:           movx a, @dptr
                mov sbuf, a
                add a, 31h
                mov 31h,a
                inc dptr
wat4:           jnb ti, wat4
                clr ti
                djnz r1, sio6
                mov a, 31h
                mov sbuf, a
wat5:           jnb ti, wat5
                clr ti
back:           pop acc
                pop psw
                reti
```

第六章　自己动手组建局域网

6.1　计算机网络

6.1.1　网络的概念

随着 VLSI 超大规模的技术和微型计算机的不断发展，计算机技术在办公自动化和工厂自动化方面的应用日益扩展，对于早先数据传输方式中，采用的以两点间一对一的传输形势，已经不能满足当前以计算机为中心的信息系统逐渐复杂化的情况，因此开始使用网络传输。

网络是由许多从一个中心服务器(server)存取文件的计算机群所组成，而每个计算机执行它自己的文件处理。网络又被认为是分布式的处理系统(distributed processing system)，因为每个系统可在自己的内存中装载和运行程序。而在分布式系统中的各个计算机通常叫做节点(node)或工作站(workstation)。在网络中被称作文件服务器(server)的装置则是用来专门处理文件存储和检索、网络管理任务、用户管理和安全性事务的。每一台计算机都需要注册进入该服务器，以便访问程序、文件和其他网络服务。

6.1.2　网络的类型

网络大致可分为三类：

(1)局域网(LAN)：通常属于一个机构的一座建筑物或一组建筑物中。

(2)城域网(MAN)：在特定的区域(如：校园、工业、城市)中的 LAN 的互连网络的集合。

(3)广域网(WAN)：是一个跨越国家和全球的网络。

局域网是组网的基本模式，其传输率目前所确认的范围为 10Mbps～100Mbps，跨距可达 100km。当前最流行的 LAN 产品为以态网(Ethernet)、令牌环网(TOKEN—RING)以及光纤分布式数据接口(FDDI)。

广域网技术及产品近 10 年来，发展迅速。目前国外数十兆传输率的广域网产品已相当成熟了。而国内使用的广域网则处于公用交换电话网(PSTN)及 X·25 公用数据网(PDN)阶段，传输率≤64kbps。具有 IP 协议的 Internet 是最流行的国际性的广域网，单机或各类网络均可联入 Internet。

6.1.3　网络拓扑结构

拓扑结构是一种研究与大小、形状无关的线和面特征的方法，图是由线所连接的点的集合。在计算机网络中，计算机作为节点，用通信线连接构成的图形称为网络的拓扑结构。局域网一般采用四种基本的拓扑结构：总线型、星型、环型和树型。如图 6.1 所示。

(1)星型：没有公用布线，所以不具备公用布线的优点。但其连接方式简单，特别是能用

电话线路传输时，可以借用电话网络，这是它的最大特点。

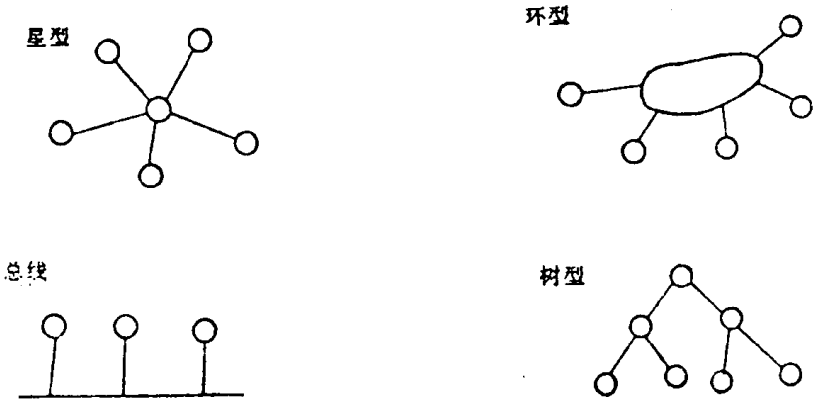


图 6.1　基本拓扑结构

(2)环型：这种方式节点之间接成闭合的环，数据在环上高速单向传送。每个节点按位传送所经过的信息。

(3)总线型：这种方式完全采用共用布线。它是把连网的计算机分别连接到通信线路的不同分支处，通信线路称为共享总线。结点的接收器随时接收总线上的信息，只有发送结点的发送器才与总线接通，向总线发送信息。

(4)树型：这种方式是将网络上的各节点按照不同的级别分层次连接。愈靠近树根，节点上的处理能力就越强。

6.1.4　建网的主要原因

谈起网络，人们可能会联想到网路上信息是如何传输与控制的，也会联想到网络上的资源是否可公用等问题。作为网络，它就是要解决多个用户同时共享网络资源，如：网络上的公用文件、网络打印机、绘图仪等。就是利用网络对网络上的用户集中管理、控制、使用，通过网络可缩小地域，扩大信息传播范围，达到高速传递信息和文档的作用。因此，建网的主要原因可包含下列几个部分：

(1)程序文件共享。

(2)网络资源共享。

(3)基于 PC 机经济性扩展。

(4)使用网络软件的能力。

(5)电子邮件。

(6)创造工作组。

(7)集中式管理。

(8)安全性。

(9)访问其他操作系统。

(10)协同结构的增强。

(11)分布式应用

6.2 网络硬件及操作系统的选择

6.2.1 局域网络操作系统简介

目前，网络操作系统类型很多，当前市场上最流行的几种网络操作系统有 Microsoft Windows NT，Novell Netware，os/2 LAN sever 及作为网络基础的 UNIX 多用户操作系统，Banyan 公司的 VINES 系统。如今，随着网络系统的不断发展，计算机网络操作系统的发展将向着能支持多种通讯协议，多种网络传输协议，支持多种网络适配器和工作站的方向发展。

1. Microsoft Windows NT

美国 Microsoft 公司着重设计 Microsoft 的新技术(NT)操作系统，到 1993 年推出了一种 32 位网络操作系统 Windows NT，它是为了满足高档，单用户桌上工作站平台，满足局域网络服务器或主干计算机系统的需要，它也是一种面向分布式图形应用程序的完整的平台系统，可在 Intel 386，486 和 Prentium 系统，以及支持 DEC 公司的 64 位的 Alpha AXP 系统。Windows NT 沿用了 windows 图形用户界面，内置网络，支持 WIN32 API 的操作系统。Windows NT 操作系统推出的版本有 Windows NT3. 1，Windows NT3. 5。

2. Novell Netware

美国 Novell 公司 1983 年推出局域网络操作系统。该公司充分吸收 UNIX 多用户，多任务的设计思想，推出了 Netware 操作系统，并实现了开放系统体系结构(OSA)从而使得 Netware 网络操作系统成为世界 LAN 网络的霸主。

Netware 网络版本可支持 30 多种网络产品，其中包括 Ethernet，Token Ring，Arcnet，PALNET 等。对于 Novell 公司它本身推出基于 80286 的 5 种网络操作系统产品，如：简易版本 ELS Ⅰ，ELS Ⅱ、高级版本 Advanced、容错版本 SFT、386 版本。前面四种版本又统称 Netware 286 版本，可以在 80286，80386，80486 上运行。而 Netware386 版只能在 80386 以上的机器上运行，对 Netware386 版有 Netwarev3. 11，V3. 12，V4. 0，V4. 1 版本。Netware386 版是功能很强的操作系统，它含有 SFT(System Fault Tolerant)系统容错技术，包括事务跟踪系统 TTS，磁盘镜象，修复重定向区，磁盘双重化，双目录和双 FAT 结构。该版本运行在 80386，80486 机上，最多可允许 250 个用户，同时打开 10 万个文件，支持 32768MB 的外存空间。Netware386 增添了口令加密措施，用户可以很方便地修改自己的口令。网络系统支持多台打印服务器，可以在工作站上实现共享打印。共享打印方式可按先进先出方式打印或按作业优先级打印。

6.2.2 网络操作系统的选择

1. 基本技术因素

首先，要了解本身建网的规模和用途，然后，根据市场上推出的各种类型的网络操作系统情况决定你所要选的网络操作系统。

网络操作系统，有简易型版本、也有高级的，容错系统 SFT 和 386 版本等等。支持的网络系统从简单到复杂，规模从小到大，适应不同的应用环境，满足不同的用户的需要。而且，

网络操作系统的技术问题多种多样，那么你可能就需要考虑适应自己的网络的拓扑结构，网络服务器的支持，网络的站点访问，网络的连接设备的支持，网络内部桥接方式，工作站的内存占有，特殊的磁盘格式和磁盘高速缓存，网络的容错功能，网络的管理，网络的安全性基于服务器的应用，电子邮件，网络的打印功能等等问题。

2. 兼容性

对于繁多的网络操作系统，它们各个都有其自身的特点和应用范围。在选择网络操作系统时，就需要考虑你所选的网络操作系统是否适合现有的设备，现有的微机中安装的操作系统。还要考虑所选择网络操作系统的潜在发展，避免引入较过时的网络版本。

我国现有的应用主要是基于 DOS 的应用，因此，在考虑网络操作系统的选择时应考虑现有 DOS 的应用是否能不加修改或基本不加修改地应用于所选择的网络系统之上，良好的和其他网络操作系统的兼容性也是能获得众多的应用软件和支持现有应用的一个重要因素。例如：Novell 的 Netware 和 Microsoft 的 Windows NT 是建立在 DOS 基础之上的，Microsoft 的 LAN Manager 是建立在 os/2 基础之上的，AT&T 的 star LAN 网络的操作系统 StarGroup 是建立在 UNIX 基础之上的。LAN Manager 和 StarGroup 对于 UNIX 系统都有良好的支持，但缺少对 Macintosh 系统的支持；而 Novell 的 Netware 则可支持 DOS、Windows、UNIX、os/2 和 Macintosh 系统。对于我国网络用户而言，应着重考虑的又一个因素是汉字应用软件的支持能力，因为我们日常处理的文字毕竟是汉字。

6.2.3 怎样选择网络硬件

选购网络时，一个很重要的因素就是网络硬件的选择。当你确定了网络操作系统后，就需要对该网络操作系统下存在有哪些网络硬件产品有所了解。同样还要考虑网络的硬件设备与你现有机器硬件兼容的问题，如 APPLE 公司的 Macintosh 和 IBM 兼容机等；网络连接设备 如网卡(Network Interface Card)，网桥(Bridge)，路由器(Router)、网关(Gateway)，中断器(Repeater)，集线器(Hub)，FDDI(Fifer Distrifuted Data Interface)等。下面简介几种网络硬件的技术指标及网卡：

IBM TOKEN R/NG 网络：该网具有极强的兼容性，网间互连性及扩展性。其主要技术指标有：

(1)网络拓扑结构是物理连线呈星形，但逻辑上呈环形。

(2)传输控制方式是令牌环访问方式，符合 IEEE802.5 标准。

(3)传输速率是 4Mbps(双绞线)，16Mbps(屏蔽双绞线)，1000Mbps(光纤)。

(4)传输介质：IBM 电缆系统目前有四种类型：TYPE1，2 是屏蔽双绞线，TYPE3 是普通双绞线，TYPE5 是光纤。

(5)网内工作站数：采用 TYPE1，2 可达 260 个工作站，采用 TYPE3 可达 72 个工作站，采用 TYPE5 可达 1000 个工作站。

IBM TOKEN RING 适配器，IBM 为不同类型的微机提供了不同的网络适配器。IBM 令牌环 PC 适配器和令牌环 161 适配器。适配器的主要功能：TOKEN 的识别与生成帧格式的构成识别和传输，传输错误检测等功能。另外，服务器可选 PS/2-80，IBM386 以上兼容机，工作站

可选 IBM-PC,PC/AT,PS/2 等。

PLAN 网络:(PERSONAL LOCAL ARER NETWORK)是美国 NESTAR 公司的局域网络产品。其网络技术指标为:

(1)网络的拓扑结构为星形。

(2)传输介质可用同轴电缆,双绞线,IBM 电缆,光纤及它们的组合。

(3)介质访问控制方式为 TOKEN PASSING。

(4)网络传输速率 2.5Mbps。

(5)最远两个工作站之间的最大距离为 6700m。

(6)最大联网工作站数 255 个。

PLAN 网络文件服务器主机采用 68000CPU。

PLAN 网络用 ARCNET 适配器,该适配器完成 IEEE802.4 数据链路层,物理层的大部分功能。市场上的 ARCNET 适配器有:PC120,PC210,PC250,PC260 等。

Novell 网络:是美国 NOVELL 公司开发的局域网络产品,它的 NETWARE 是当今世界上最为流行的网络操作系统之一。该系统可支持包括 3COM ETHERNET 网络,IBM TOKENR/NG 网络,IBM PCNET 网络及 ARCNET 网络等著名网络产品。

NOVELL 以太网络硬件组网的技术指标:

(1)网络拓扑结构为总线形。

(2)传输控制方式为 CSMA/CD。

(3)传输介质为 50Ω 粗、细同轴电缆。

(4)传输速度为 10Mbps。

(5)网内站间最大距离为 250m。

(6)网段工作站间最大距离为 500m(粗缆),185m(细缆)。

(7)网内最多工作站数为 1024 个。

(8)网段最多工作数为 100 个(粗缆),30 个(细缆)。

(9)收发器之间最小距离 2.5m(粗缆),0.5m(细缆)。

Novell 的 NETWARE 有两种流行的版本,NETWARE286 2·15 版,NETWARE386 3·11 版、前者所有服务器必须是 IBM PC 286 以上机。后者其服务器必须是 INTEL 386 系列以上的微机。

Novell 的网络适配器有 NE 1000 卡,数据总线为 8 位,适用于 IBM-PC/XT,PC/AT 及兼容机。NE 2000 卡,数据总线为 16 位,适用于 IBM PC/AT 及兼容机。NE 3200 卡,数据总线为 32 位,适用于 INTEL 80386,80486 及系列兼容机。

6.3 自己动手组建局域网络硬件系统

6.3.1 网络硬件基础

一、基本硬件

网络硬件包括:文件服务器,工作站,网络适配器,BNC 连接器及终结器,同轴电缆。

二、连结步骤

1. 文件服务器，工作站

若选择以太网，则工作站可用 IBM-PC 系列及兼容机，文件服务器可选 INTEL286，386 等系列微机。内存 4M 以上，硬盘在 300M 以上。

2. 适配器

适配器是一块插件板，也称 NIC(Network Interface Card)网卡，它插接在工作站、文件服务器扩展槽上。适配器根据网络协议，通过同轴电缆等连线，进行数据包的发送和接收等等功能。另外，网卡本身有两种硬件情况，一种是有跳线卡，一种是无跳线卡。在连接卡时若是无跳线卡，则需安装软件驱动程序进行相应的设置。若有跳线则可根据图，按具体情况加跳线来进行设置。即 I/O 基址选择，中断号选择，内存地址选择，DMA 选择等。

3. 传输介质

(1)双绞线：它可用于传输数字信号和模拟信号。其成本低，辅设简单。但易受干扰，传输效率低等等。

(2)同轴电缆：它具有很强的抗干扰能力，带宽高、吞吐量大、安装方便。它有两种基本型：基带同轴电缆和宽带同轴电缆。基带同轴电缆通常用屏蔽线用铜做成网状，特征阻抗为 50Ω，用于数字信号传输。宽带同轴电缆其屏蔽层通常是铝冲压形成的，特征阻抗 75Ω，用于模拟信号的传输。

(3)光纤：利用光纤通信有频带极高，衰减极低，具有很强的抗干扰性和比普通同轴电缆通信容量大 100 倍左右等优点。但其连接光缆技术要求高。

4. 线间连接器

工作站与干线，干线与干线，文件服务器与工作干线，它们之间都是通过连接器连接的。做为网络的连接器有：BNC 接头、BNC 管形接头、T 形接头等，RJ-45 双绞线接头、插座、AUI 插座。如图 6.2 所示。

6.3.2 有盘站和无盘工作站

假如，你连接的是有盘工作站，则在网卡上可不用 EPROM 启动芯片，而是在工作站的硬盘上建立网络启动程序。此时，工作站应该有相应的操作系统存在，如 DOS 系统。用网络工作站启动软件生成启动程序。

若你是无盘工作站(即无硬驱动)这时，在网卡上就需要有一个网络启动芯卡(EPROM)，EPROM 上有一启动文件。如果每一台无盘工作站使用一块不同的卡，就必须决定该卡的地址并为它创建一个不同的启动文件。将这些地址记入服务器上的一个注册文件中，以表明哪一台工作站将使用哪一个特殊的启动文件。

6.3.3 实例

若某单位有 20 台 IBM386，一台 486 机，想建立一个局域网，应如何选择硬件系统？

根据单位现有微机，若要连接一个价廉，实用性能稳定的局域网络系统，可按下列步骤

进行：

1. 选择网卡

选择网卡时，可根据市场流行的产品挑选。如 Novell 的 NE2000 系统的兼容网卡，还有如，3COM，Acctond-Link 网卡系列。

假如：你选择了，NE2000 兼容的带 EPROM 的网卡，那么，首先要将网卡的跳线连接好，即①将 JPL 或 JP 选择选到 JPL 上。②将 I/O BASE 跳线选择插座，用短接跳线设置成 4，5，6 座为 OFF 状态。其中的 I/O IRQ1，2，3 管座也选成 OFF 状态。③存储器地址选择有几种，你可根据机器使用口地址的情况来决定网卡存储器地址的选择，如选 D800 则在 MEM 接座上将 1，2，3 接成 OFF 位置，4 接成 on 位置。

若你选择的是无跳线网卡，则此网卡可直接插入计算机扩展槽上，其 I/O BASE，IRQ MEM 等的设置由网络卡的驱动软件来设置。

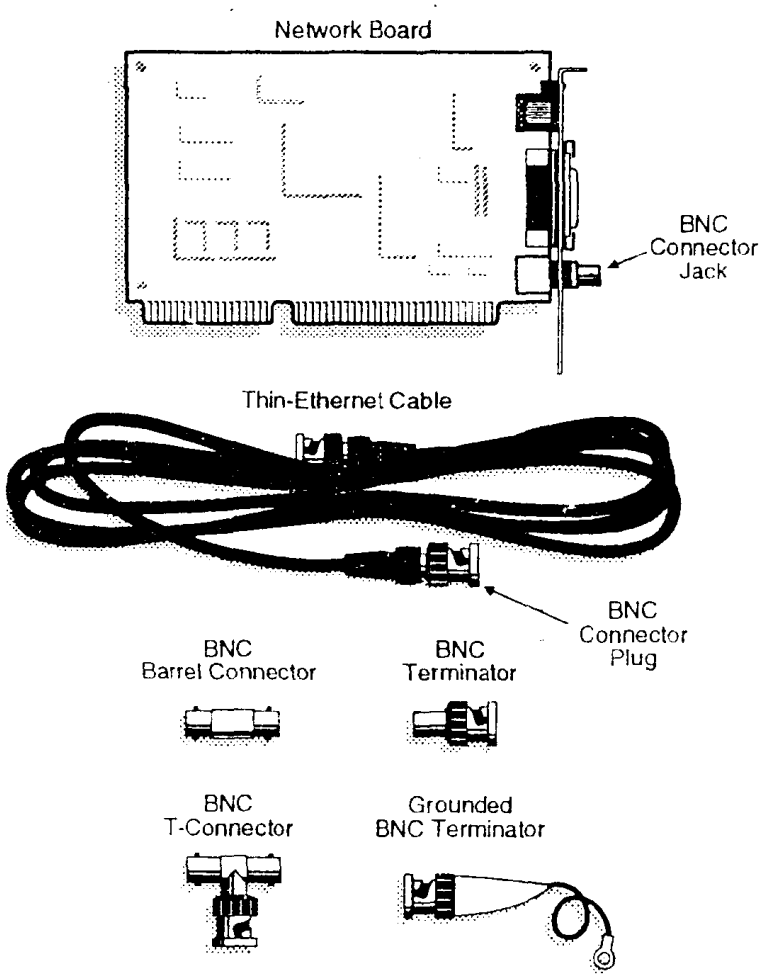


图 6.2　网络连接器

2. 选择连线

根据所选网络的拓扑结构，网络的网卡等，决定连线的类型。一般情况，一个小型局域网，若采用广播信道网络中总线结构，连线可选同轴电缆。而同轴电缆一般指细同轴电缆，这种电缆带宽大约接近 100MHz。距离为 2500 米，抗强电干扰性能高，安装容易，布局多样，价

格低廉，保密性好。

3. 选择连接器

根据所选网卡的插座而定(注：有的网卡上同时装有 BNC，RJ-45，AUI 三种连接插座)，通常选用 BNC 插头，这种连接器适合于同轴电缆线连结，且装卸方便，但也易出现接触问题。若选 RJ-45 连接器则需选 UTP(Unshielded Twisted-Pair)连线。RJ-45 接线较麻烦抗干扰性能差，但其连接头，座间不易出现接触问题，连线间接触良好，不易发生因连接器问题而产生的网络死机现象。

4. 终结器

终结器是用于网络连接时，网络的匹配。终结器有 50Ω，75Ω 两种，若采用细同轴电缆线则可选 N 系列 50Ω 终结器，N 系列终结器用于 BNC 连接头，此连接器上带有地线连接端。

整个硬件的连接如图 6.3。WS 代表工作站。

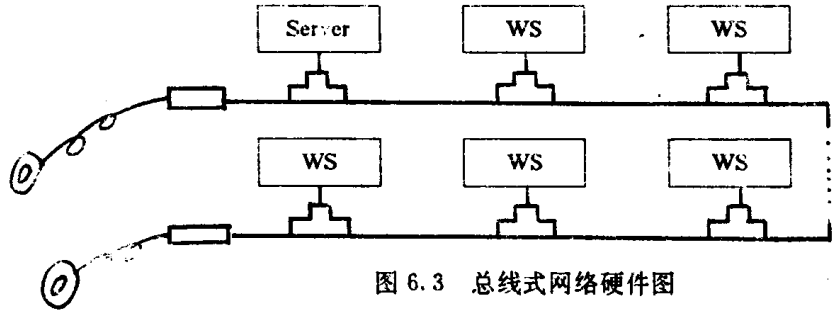


图 6.3　总线式网络硬件图

6.4　自己动手安装 NOVELL 网络操作系统

本节以 NETWARE V3.12 版本为例讨论 NOVELL 网络操作系统的安装。

6.4.1　网络协议

安装网络操作系统时，必须了解网络协议，通过网络协议可决定你所联网的物理结构和所选网络的类型等。下面对网络协议作一简介。

局域网协议标准化的工作，由世界上著名的标准化组织来决定的，如：美国电气电子工程师协会(IEEE)的 IEEE802，国际标准化组织(ISO)的 ISO8802 和国际电工委员会(IEC)的 PROWAY。

在局域网标准的研究中，美国电气电子工程师协会处于领先地位，该协会从 80 年代开始推出了六项标准文本，其内容如下：

(1)IEEE802.1：系统结构和网际互连。

(2)IEEE802.2：逻辑链路控制。

(3)IEEE802.3：CSMA/CD 总线访问方法和物理层技术规范。

(4)IEEE802.4：Token passing Bus 访问方法和物理层技术规范。

(5)IEEE802.5：Token passing Ring 访问方法和物理层技术规范。

(6)IEEE802.6：城市网络访问方法和物理层技术规范。

这里以目前国内使用较多的 IEEE802.3 标准作为实例说明如下：

IEEE802.3 CSMA/CD 是一个使用 CSMA/CD 访问方法的局域网的综合性标准。它详细说明了总线拓扑结构的 CSMA/CD 介质访问控制 MAC 子层的服务规范、帧结构格式、控制方式的功能。而 CSMA 为载波监听多路访问(Carier Sense MultipleAccess)，它的作用是将想要传输的站首先对信道进行监听以确定是否有别的站在传输。如果信道空闲，可以传输，否则该站根据一定的算法策略退避一段时间再重新监听、传送。在 CSMA 上加上 CD(Collision Detection)冲突检测功能，这时站上可随时检测到冲突的发生，减少带宽的浪费，提高系统的效率。对于这种连接也有局限性、因这种局域网在负载不重的情况时，才有较好的性能(较短的获取信道延迟等待时间)，但在负载很重时将由于大量碰撞而损失信道容量。若反复碰撞，信道获取时间无限延长，没有确定的上界。在 IEEE802.3 中规定的媒体为 50Ω 的基带同轴电缆，数据速率为 10Mb/s，最大段长为 500m，简记为 10BASE5。信号采用曼彻斯特(manchester)编码。

6.4.2 文件服务器的安装

1. 用 DOS3.31 以上版本启动服务器主机。

2. 载入 Install 盘以执行 NETWARE 低级格式化或删除旧分区(可选)。

3. 用 FDISK 在硬盘上建立一个 DOS 分区、并激活(Active)分区。建议选 10MB 的 DOS 分区。

4. 格式化新建 DOS 分区。

FORMAT C:/s〈Enter〉

5. INSTALL 盘

(1)键入 INSTALL 〈Enter〉

(2)选提示菜单中的“install new Netware 3.12”。回车

(3)选提示菜单中的“Retain current disk partition”回车

(4)提示菜单问服务器名，你可在光标处输入其名字如：键入 FWQ〈Enter〉

(5)提示菜单问内部网络号，可在光标处输入号码，如：键入 00006668〈Enter〉(用 Del 键可删除原来的号码)。

(6)第一张盘安装结束，屏幕上出现 DOS 的提示光标。提示插入 SYSTEM-1 盘。

6. 放入 SYSTEM-1 盘。

(1)回车开始拷贝该盘。

(2)根据提示插入 SYSTEM-2，回车开始拷贝该盘。

7. 放入 UNICODE 盘。

(1)屏幕提示选择国家代码、代码页数字：

Press〈Enter〉 to view choices
country code 001 (united states)
code pate:437(united states English)
keyboard Mapping:None

若选缺省值则敲 F10 键(Accept local setlings)。

(2)提示选择文件格式

select the format you desire and press〈Enter〉
Dos Filename Format (recommended)
Netware Filename Format

建议选"Dos Filename Format"格式。

(3)你想设置启动命令否

Do you want to specify any special(startupsetcom mand:)
No
Yes

建议选"No"。

(4)提示装 server 文件列 Autoexec. BAT 文件中否:

Do you want Autoexec. bat to load server. EXE?
No
Yes

建议选择"Yes"。

(5)提示输入批处理的路径:

Enter the path to your Autoexec. bat file
path: c: \ Autoexec. bat_

若选缺省值,可敲回车。

(6)到此,SYSTSM-1,2,UNICODE 盘基本功能安装完毕,屏幕显示出带服务器名的光标:

FWQ:_

8. 安装硬盘驱动程序并承认其默认参数值。如果文件服务器主机为标准的 AT 总线结构,则装入 ISADISK 硬盘驱动程序。如:

FWQ:LOAD ISADISK 〈Enter〉(此文件通过上述过程已装入硬盘)

默认值是 Port=1F0 INT=E。

9. 装入 Install 模块

LOAD Install 〈Enter〉

(1)显示:安装选择菜单

installation Option
Disk option
volume option
system option
product option
EXIT

若选择“Disk option”。

(2)显示磁盘选择信息窗口

Available Disk Option
Format (optional)
Partition tables
Mirroring
Surfare test (option)
Return to main menu

建立分区表:选择“Partition tables”〈Enter〉

(3)显示分区表菜单窗口

Partition Options
Change hot fix
Create Netware partition
Delete partion
Return to previous menu

根据具体情况选择、若是第一次安装,则选“Create Netware partition”建立 Netware 分区。若是重新安装 Netware,则先选“Delete partion”,删除原分区,再选“Create Netware partition”建立 Netware 分区。

(4)选择安装选择窗口中的“Volume option”显示:

New Volume Information
Volume Name : SYS
Volume Block Size :4K Blocks
Initial Segment Size : 1225/2 Blocks
Volume Size : 490Meg
Status : Not Mounted

在此窗口中，敲回车，提示建立新卷标 sys 否，回答选“Yes”。建立完卷标后，返回到“New Volume Information”窗口，选择“Status”项，对 sys 卷进行安装(mount)。

10. 进入“Installation Optien”窗中的“system Option”显示系统选择窗口：

Available System Options
Copy System and public files
Create Autoexec. NCF files
Create startup. NCF files
Edit Autoexec. NCF files
Edit startup. NCF files
Return to main menu

此窗口开始选择安装 system 和 public 文件选“Copy system and public files”根据提示安装以下磁盘：

Install 盘，system-1，2 盘，UNICODE 盘，system-3，4，5，6，7，8 系统盘。

11. 根据连结的电缆和网卡，装入相应的 LAN 驱动程序，如：

系统提示选 I/O 基地址，中断号等硬件参数，一般选择 I/O BASE=300，IRQ=3

12. 将指定的核心协议连接到网卡驱动程序上。

BIND〈协议模块〉to〈网卡驱动程序〉

一般选：

FWQ:BIND IPX to NE2000 〈Enter〉

13. 再次装入 Install 模块，建立服务器启动文件：

FWQ:LOAD INSTALL 〈Enter〉

进入“Installation Option”中的“System Options”项、选择“Autoexec. NCF”和“Startup. NCF”分别建立启动文件。

系统自动生成的 Autoexec. NCF 文件为：

file server name FWQ

ipx internal net 6668

Load NE2000 port=300 INT=3 frame=ETHERNET-802. 2

Bind IPX to NE2000 net=4964

startup. NCF 内容为：

Load ISADISK port=1F0 int=E

通过以上 13 个步骤，详细地介绍了 Netware386 文件服务器的安装，配置的过程。安装了文件服务器后，每次启动时，系统即通过执行 SERVER. EXE，STARTUP. NCF，AUTOEXEC. NC 文件启动，进入文件服务器主控台方式。

6.4.3 工作站的安装

工作站的安装有两种情况：有盘站和无盘站。

1. 有盘站的安装

(1)用 V3. 12 的 WSDOS-1,WSDRV-2 盘安装(注:若用 V3. 11 版,则用 WSGEN,WSDRV-1,2 盘安装)

首先插入 WSDOS-1 盘到 A 驱动器键入:

A:>INSTALL 〈Enter〉

(2)屏幕出现提示,问是否建立 NWCLIENT 子目录,若要建立,回答时敲两次〈Enter〉键。接着问是否要自动地在 Autoexec. bat 文件中建此子目录及启动文件,若要建立,则敲入"Yes"后回车,屏幕上的其他提示都以默认值回答。

(3)根据提示插入 WSDRV-2 盘,回车键进入。屏幕上提示出是否要输入或修改网卡类型、基地址、中断号、协议、节点号和存储地址。这时可按具体硬件情况选择参数。

例如:选①网卡为 NE2000 系列的网卡类型值 NE2000

② BASE I/O PORT=300(选择修改后,敲"F5"键确定)

③ INT=3

④ FRAME Ethernet_802. 3

⑤ MEM=D800

⑥ node address=485445001174

参数设置后,敲"Esc"键确认。

(4)根据提示插入 WSDOS-1 〈Enter〉

这步开始拷贝 WSDOS-1 的文件。拷贝完后提示建立或修改 NET. CFG 和 STARTNET. BAT 文件,原来的 Autoexec. bat 等启动文件被拷贝进入以 BNW 为扩展名的文件中去了。敲〈Enter〉键退出安装。敲 ALT+Ctrl+Del 键,重新启动工作站。

(5)以上三步建立了一个有盘工作站,在重新启动系统,将工作站与服务器联接起来,并用超级用户名"Supervisor"注入登录,系统会显示:

```
DRIVE A:mAPS to a local Disk
DRIVE B:mAPS to a local Disk
DRIVE C:mAPS to a local Disk
DRIVE D:mAPS to a local Disk
DRIVE F:=FWQ\SYS:\LOGIN
SEARCH 1:=Z:[FWQ\SYS:\PUBLIC]
SEARCH 2:=Y:[FWQ\SYS:\]
```

2. 无盘站的安装

无盘工作站是通过 Boot ROM 芯片上的文件与文件服务器连接,把文件共享盘的 Netware 分区 sys 卷的 Login 目录中的伪盘文件(NET $ DOS.SYS)看成无盘工作站的 A 盘。从它引导 DOS 并执行 autoexec. bat 中的各条命令,这些命令包含加载工作站 shell 程序,并把当前驱动器转向下一个网络驱动器 F:。下面将介绍无盘站的安装步骤:

(1)格式化一张软盘,使其能自动启动 DOS 系统,并且在该盘中需包含 Netware 的 shell 程序使该盘能自动加载 shell,自动转到第一个网络驱动器上。

(2)插入启动盘到 A 驱动器中,打开电源开关,进入引导过程(服务器必须打开,且网卡,连线等已经准备完备),见屏幕提示输入:

SUPERVISOR 〈Enter〉

(3)在工作站上,把驱动器 G:映射到 SYSTEM。如

MAP G:=SYS:\SYSTEM

(4)用 MAP 查看映射路径:如

MAP 〈Enter〉

显:

DRIVE F:=SYS:\LOGIN

DRIVE G:=SYS:\SYSTEM

(5)把当前驱动器转为 G:

(6)运行 DOS 远程映象文件生成程序

G:>DOSGEN 〈Enter〉

此时,把引导文件盘上的各种文件处理后,归入伪文件 NET$DOS.SYS,此文件要建立在 SYS 卷 LOGIN 目录中。

(7)把引导盘上的 Autoexec.bat 拷贝到 SYS:LOGIN 目录。

(8)将生成的伪盘文件标志为可共享:

flag NET$ DOS.sys s〈Enter〉

flag autoexec.bat s〈Enter〉

通过以上八个步骤,无盘工作站的连接工作安装完毕。余下来的工作就是要对网络进行管理、建立必要的一些文件如注册正本,对网络进行网络维护,保证网络的正常运转。

6.5 NOVELL 网络系统的管理

网络连接完毕后,就要投入实际使用,然而,要想很好的使用网络系统,则对其实用程序的应用,网络软、硬件的管理方面就必须要有一定的了解,成功的网络管理不仅确保网络正常运行,而且确保网络有效地实现功能。

6.5.1 Netware 目录结构

Netware 目录结构类似于 DOS 的目录结构,它也是采用的树型结构,树根为文件服务器,文件服务器下又组成几个文件卷、文件卷又可以组成若干个目录,目录下有子目录和文件,子目录下又可分子目录和文件。文件服务器启动之后,系统自动创建 sys 卷下的 4 个子目录:

(1)sys:LOGIN 目录,该目录用于工作站注册所必须的程序。

(2)sys:MAIL 目录,该目录由与 Netware 兼容 的邮件程序所使用,另外还兼作用户注册正本和打印作业配置文件的存储。

(3)sys:PUBLIC 目录,该目录包含 Netware 实用程序和为网络用户准备的程序。

(4)sys:SYSTEM 目录,该目录包含操作系统文件及 Netware 实用程序和为网络管理员准备的程序。

(5)目录的组建,可用实用程序 SYSCON.EXE 建立。

(6)目录设计实例如图 6.1。

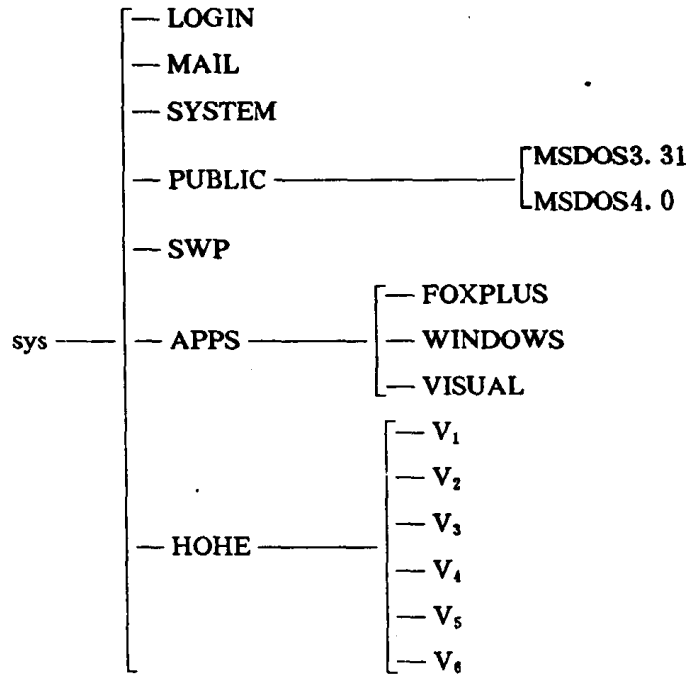


图 6.4　目录设计实例

6.5.2　网络驱动器类型

网络驱动器分为：{本地驱动器、网络驱动器、检索驱动器}

(1)本地驱动器：是指工作站上的软盘驱动器，硬盘驱动器。

(2)网络驱动器：是指文件服务器上的卷内的一个目录与驱动器标识符映像得到的驱动器。

(3)检索驱动器：是指能提供搜索非当前目录的文件的驱动器。它也是网络驱动器中间的一种。

Netware 驱动器的规定：

(1)英文字母 A 到 Z 都可以作网络驱动器。

(2)本地驱动器所用字母一般为 A、B、C、D。

(3)检索驱动器所用字母为 K～Z。且计算机网络的检索驱动器字母是从 Z 开始使用的。

(4)Netware 规定 F 字母为第一个网络驱动器，然后依次为 G，H 一直到 Z。

(5)在设置网络驱动器和检索驱动器时，要注意防止发生冲突。

(6)网络上驱动器的映射由 MAP 映像命令来完成。

6.5.3　Netware 用户和组的概念

用户是指使用和管理网络资源的人。

用户组是指能够使用相同的应用程序，执行相同任务或者有相似的信息或打印需要的用户。

. Netware 网络生成后，在网上的目录中，我们能看到两个用户 supervisor 和 GUEST 和一个用户组 everyone。而前两个用户又属于 everyone 组中的成员。我们在第一次启动系统时（指刚安装完系统），就可用这两个用户名进入网络，对网络进行管理操作。

Netware 网用户分为三种类型：

用户 { 普通用户 / 网络操作员 / 系统管理员

(1)普通用户是指仅能使用网络资源和自己的软件的用户。

(2)网络操作员是指具有部分管理网络资源和用户的权力，并且同时也具备一般性用户的所有权限。

(3)系统管理员是指具有对整个网络资源及用户有管理权限的成员。它是具有最高的管理权限，可以完成所有的网络操作。如它可以作为用户组的管理员。

(4)用户及用户组的建立，可用 syscon. exe 文件通过网络操作员，系统管理员来完成。

6.5.4 Netware 的注册正本

Netware 注册正本(LOGIN SCRIPT)的功能同 DOS 中的自启动批处理文件很相似，它包含了用户注册入网时要执行的各种命令，建立入网后的网络环境。用户注册时按顺序一条一条地执行注册正本中的命令。

一般情况下，注册正本包含驱动器和检索驱动器的映象，也可包含一些问候信息，连接其他服务器，显示选择菜单等。Netware 的注册正本有三种类型：

注册正本 { 默认注册正本 / 系统注册正本 / 用户注册正本

(1)默认注册正本：它是用在当用户没有建立自己的用户注册正本时，就调默认注册正本规定的命令，这个正本放在 LOGIN. EXE 文件中，注册正本不可修改。

(2)系统注册正本：它只有管理员有权建立和修改。这个正本是为用户设置网络参数的，它提供每个用户所需公用的信息。用户在入网时，首先进入这个正本然后才是自己工作站的用户注册正本。系统注册正本放在 SYS:PUBLIC 下的 NET $ LOG. DAT 中。

(3)用户注册正本：它是为用户入网时，执行本地命令、建立自己的信息之用的。它存放在 SYS:MAIL 的各用户标识 ID 目录下。

(4)建立注册正本可通过 SYSCON. EXE 文件来实现。

(5)建立注册正本的注意事项：

- 只有系统管理员可编辑系统注册正本。
- 用户只能编辑用户正本。
- 注册正本中的命令必须是注册正本命令若要使用 DOS 命令，须在命令前加 # 号(#)。
- 每行开头必须是注册正本命令。
- 每行可键入一条命令，每一命令行长度不可超过 150 个字符长。
- 每行结尾必须用〈Enter〉结束。

• 用 REMARK 或(分号";"以及 REM)来标注一行注释。该注释执行时不显示出来。

• 网络正本用标识符前必须用百分号"%",且标识符本身要大写。标识符见表 6-1 所示。

表 6-1 标识符

序号	标识符	含义及返回值
1	YEAR	年号　取值为(1994,等)
2	MONTH	月号　取值为(01～12)
3	DAY	日号　取值为(01～31)
4	AM—PM	返回值为 am 或 pm,上下午
5	HOUR	当前小时取值为(1～12)
6	HOUR24	以 24 小时取值,返回值为(00～23)
7	MINUTE	当前分钟数　取值为(00～59)
8	SECOND	当前秒钟数　取值为(00～59)
9	LOGIN—NAME	当前注册的用户名
10	FULL—NAME	当前注册的用户全名,可用 47 个字母
11	STATION	当前用户注册的键接号
12	P—STATION	当前工作站的节点地址,取 12 位十六进制数
13	GREETING—TIME	问候时的取值为 morning,afternoon…
14	OS	返回工作站操作系统类型
15	ERROR—LEVEL	返回数值,指示错误类型,0 表无错
16	FILE SERVER	返回当前文件服务器名

(6)注册正本命令

注册正本命令常用的有 DRIVE,MAP,WRITE,DISPLAY,PAUSE,#,COMSPEC 等等。

a. DRIVE 命令

它是设置当前默认的驱动器号。

格式:DRIVE　n:/*m:

n 为驱动器号,m 为第 m 个网络驱动器(第 1 个通常是 F:)它用来改变当前默认的驱动器号。

b. MAP 命令

该命令处理逻辑和检索驱动器映射如:

• 建立一个网络驱动器映射。

MAP　Z:=SYS:PUBLIC　〈Enter〉

• 用 INS 插入驱动器到个人

MAP　INS　S16:=SYS:PUBLIC\FOX

• 用 MAP 命令将一驱动器映射到一个虚根目录,如将 QT 目录映像成一个虚根目录。

MAP　ROOT F:=SYS:APPS\WIN\QT

• MAP DEL[驱动器号]为删除指定驱动器

• MAP DISPLAY on 入网时显示映像关系,该命令默认有效。

• MAP DISPLAY OFF 入网时不显示映像关系。

• MAP ERROR on 入网时显示错误信息。

• MAP ERROR OFF 入网时不显示错误信息。

c. PAUSE 命令

格式 PAUSE

暂停注册正本命令的执行,按任意键继续。

d. WRITE 命令

格式:WRITE"text";标识参量。

或 WRITE"text%标识参量"

它的用途是在屏幕上显示信息。可有多条信息出现,这时它们之间用分号(;)隔开。如:

WRITE "GOOD % GREETING-TIME","Today is % DAY"

e. #号命令

它是用来执行一个外部程序的。

格式:#[Path/] 命令名 [命令参数]

执行的外部参数可以是 DOS 的内部命令、外部命令(.EXE,.COM 程序)和批处理程序(.BAT),外部命令执行完后,返回注册正本命令处理。

f. COMSPEC 命令

格式:COMSPEC[Path/*n:/sn:\]COMMAND.COM

该命令告诉 Netware 在何处找 DOS 的 COMMAND.COM 程序或 DOS 外壳。*n 中的 n 表示第 n 个网络驱动器,Sn 表示第 n 个检索驱动器。

如:COMSPEC=NETSERVER/SYS:DOS\COMMAND.COM

6.5.5 网络基本操作

一、入网、退网命令

1. 入网

LOGIN 〈Enter〉

然后输入用户登录名和口令。

2. 退网

LOGOUT 〈Enter〉

二、权限、属性操作

1. 查看、设置或修改目录或文件的继承权限标志(IRM)

格式:Allow [路径[TOINHERIT] [权限表]

Read——打开,读文件。

Write——打开,写文件。

Create——创建和写文件。

Erase——删除目录。

Modify——改变目录和文件属性;

Filescan——查看目录下的所有文件。

Access Control——修改目录和文件下的委托指定和继承权限标志(IRM)。

All——具有所有 8 种受托权限。

Supervisory——指定目录,文件具有所有权限。该项忽略所有用 IRM 加在这些目录、文件的权限。

Nothing——从继承权限标志(IRM)中去掉除 supervisory 外的其余所有权限。

如:Allow *.DAT RWEM 〈Enter〉

设置文件扩展名为 DAT 的文件有继承 Read,Write,Erase,Modify 权限。

2. 查看或改变指定目录下文件的属性

格式:FLAG [路径] [属性表]

Shareable——文件为共享。

Non-Shareadle——文件为非共享。

Read-Only——文件为仅读。

Read-Write——文件为可读写。

Normal——文件为普通文件。

Transactional——文件为事务文件。

Indexed——文件为索引文件。

如:FLAG SYS:PUBLIC\FOX*.* RWS 〈Enter〉

它将 SYS:PUBLIC\FOX 下的所有文件设置为可读写和共享方式。

3. 设置或查看目录属性

格式:FLAGDIR [路径][属性表]

Hidden——隐含,System——系统,

Private——私用,Normal——普通。

如:FLAGDIR SYS:APPS H 〈Enter〉

将设置 SYS 卷下目录 APPS 隐含属性。

三、目录、文件的查看

1. 目录操作命令

格式:NDIR [路径][参数]

参数:DO——仅对目录,SUB——对子目录及其下的文件。

OW——所有者,CR——创建的目录。

如:NDIR * DO 〈Enter〉

显示所有目录清单。

又如:NDIR SYS:SUB 〈Enter〉

显示 SYS 卷下的所有子目录和文件。

2. 查看本用户的使用情况

格式:WHOAMZ [服务器][选项]

选项:/group　用户组,/security　安全等效,

/Rights　目录权限,/All　都选。

3. 查看已登录入户的情况

格式:USERLIST　[服务器]/用户[/All]

All 表示列出连接号,登录时间,节点号,网络地址。

4. 查看子目录

格式:LISTDIR　[路径][选项]

/subdirectory　查看后继的子目录,

/Right　查看后继的最大权限,

/Date 或 Time　查创建的时期,

/All　上述所有。

6.5.6　实用程序的应用

Netware 的实用程序有很多种。如:SYS CON,PCONSOLE,MAKEUSER,FILER,SESSON 等等。Netware 网络操作系统通过这些实用程序对网络进行管理。为和户掌握 Netware 菜单实用程序,目录结构,网络用户,网络安全保护和文件组织方式等内容提供很好的帮助。下面以 SYS CON 程序为例,说明实用程序的使用方法:

一、SYS CON 程序使用

实用程序 SYS CON(系统配制管理)是由系统管理员或等效用户使用的。它是用于建立、删除用户、设定或修改用户信息、保证网上信息安全性的软件。

实用程序 SYS CON 的启动方法:

(1)首先,在工作站上键入文件名 SYS CON。

(2)进入 SYS CON 程序的菜单提示窗口如:

Available Topics
Accounting
Change Current Server
File Server Information
Group Information
Supervisor Option
User Information

在"Available Topics"主菜单中共有 6 个提示选项,每个选项又对应各自的子菜单。每个选项的意思分别为:①记帐;②改变当前服务器;③文件服务器信息;④用户组信息;⑤管理员选项;⑥用户信息。

(3)若选择"Supervisor Option"项,则显:

Supervisor Options
Defaiclt Accouiit Balance/Restrictions
Default Time Restrictions
Edit System AUTOEXEC File
File Server Console Operators
Intruder Detection/Lockout
System Login Script
View File Server Error Log

通过这个菜单管理员可对系统设置参数:①默认帐户余额或限制;②默认的时间限制;③编辑系统批文件 AUTOEXEC. BAT;④文件服务器主控制台操作员;⑤非法者检测/锁定;⑥系统入网注册正本;⑦查看文件服务器上的错误登录记录;⑧系统工作组管理者。

假如,想要修改系统批文件则可选"Edit System Autoexec File"项,选择后,屏幕将显示原批处理的内容。

4. 若选主菜单中的"USER Information"项则显:

User Information
Account Balance
Account Restrictions
Change password
Full Name
Group Belonged to
Intruder Lockout status
Login Script
Managed Users and Groups
Managers
Other Information
Security Equivalances
Station Restrictions
Time Restrictions
Trustee Directory Assignments
Trustee File Assignments
Volume Restrictions

在这个菜单中,可对用户做参数指定,生成自己的环境。它可修改:①帐号余额;②帐户限制;③设定或修改用户口令;④用户名全称,共 47 个字符;⑤所属的用户组;⑥非法者锁定状态;⑦用户入网正本;⑧被管理的用户和用户组;⑨管理者列表;⑩其他信息;⑪保密等效;⑫工作站限制;⑬时间限制;⑭目录代管权分配;⑮文件代管权分配;⑯卷限制。

假若你想插入新的目录并建立本站上的节点地址，那么可选本菜单选项，且在本菜单选项显示后，按 Ins 键进入：

User name:

在此窗口中键入目录名并回车即可、然后在本菜单中选“station Restrictions”项提示：

Allowed Login Addresses

这时敲 Ins 键，进入网络地址输入窗口：

Network Address:

可输入 8 位十六进制数 * 000004964，回车确认接着显：

Allow Login From All Nodes
No
Yes

问允许从任意节点入网吗？若回答“Yes”那该用户可从网络的任何站上入网。若回答“No”，那么屏幕出现另一窗口

Node address:

提示从这窗口处输入 12 位十六进制数，作为本站的节点地址。这个节点数由网卡硬件决定。可通过命令 USERLIST/A 获得或连接工作站时获得。

如：18543500175c，回车确认，并显示：

Allowed Login Addresses	
00004964	18543500175c

5. 增加或删除代管权限

(1)选“User Information”项中的“Trustee Directory Assignments”，进入受权者目录委派窗口，若想对 SYS:HOME\U1 增加存取控制权限，首先选择 SYS:HOME\U1 为高亮状态，并回车确认。这时屏幕出现“Trustee Rights Granted”窗口。显示出 U1 原来的权限值：

Trustee Rights Granted
File Scan
Read
Crease
Erase
Modify
Write

(2)列出“trustee Rights Granted”窗后，敲“Ins”键显出“Trustee Rights Not Granted”窗口：

Trustee Rights Not Granted
Access Control
Supervisory

(3)用编辑键,从窗口上选择"Access Control"项,回车确认。这时存取控制权限就被加在"trustee Rights Granted"窗口内,使得 U1 这个目录有了存取控制权限。如下图计算:

Trustee Rights Granted
Access Control
Crease
Erase
Modify
Read
File Scan
Write

(4)若想要删除某个权限,可在要删除权限处敲"Del"键,回答"Yes"确认即可。

二、Pconsole 实用程序

Pconsole 实用程序是用于管理打印机的打印队列的,此程序可以用来建立和重命名打印队列、指定队列用户和队列操作员。

假若需在工作站 4 号机上建立一个远程打印机 Print1,可按下列步骤进行:

1. 由管理员运行 Pconsole 程序,显示出主菜单:

Available Options
Change current file server
Print Queue Information
Print Server Information

主菜单中第一项用来改变到不同文件服务器,查看有关当前文件服务器的信息。第二项用来定义新的打印队列、增加或删除队列中的打印作业,以及删除队列。第三项用来列出服务器信息,设置队列名、口令、打印机参数等。

2. 选主菜单中"Print Server Information"项,则显示:

Print Servers

敲"Ins"键后提示:

New Print Server Name:

这时键入 FWQ1〈Enter〉,把 FWQ 送入"Print Servers"窗口中。然后再回车进入下面窗口:

Print Server Information
Change password
full name
Print server configuration
Print server ID
Print server operators
Print server status/control
Print server users

3. 选“Print Server Information”中的第三项“Print Server Configuration”，给打印服务器配置参数。并显示：

Print Server Configuration menu
File Servers to be Serviced
Notify List for Printer
Printer configuration
Queues Serviced by Printer

4. 选“Print Server Configuration menu”中的“Print Configuration”项，打印机配置栏，回车确认后显：

Configured Printers	
Not installed	0
⋮	⋮
Not installed	15

在此窗口中选择第二项“Not installed 1”回车显示 1 号机所需的配置表：

Print 1 configuration
Name:Print 1
Type:Defined elsewhere
Use interrupts:
⋮

在“Print 1 configuration”中选 Type 项，确认后屏幕打印出串口、并口等类型参数，这时选“Remote Parallel LPT1”，将“Print 1 Configuration”中的 Type 设置成“Remote Parallel LPT1”。键入 Esc 键存参数。

5. 在“Print Server Configuration menu”中选“notify list for Printer”项，显示：

Defined Printers	
Printer 0	0
Printer 1	1

在定义的打印队列中选“Print 1”回车后敲“Ins”键，显示：

Notify Candidates	
Supervisor	User
US1	User
US4	User

6. 在“Notify Candidates”中选 US4 用户，屏幕提示“Notify internal”内容“first 30”和“second 60”，敲“Esc”键确定，回答“Yes”并存储变化的参数。并且显示：

file Server	Notify name	Notify Type	First	Next
FWQ	US4	(user)	30	60

到此打印机 PRINT1 已经建立配置完成，最后根据情况配置打印机管理者名，服务器用户名单等即可。

7. Print 1 机的连接

当建立队列后，网络系统要求必须在服务器控制台上通过说明打印队列的 ADD 命令，将队列映射到打印机，或通过 AUTOEXEC 文件实现这映射，只有这样队列中的作业才能在网络上打印出来。如，将控制台命令把新队列映射到 1 号打印机上，使用 0 号缓冲区：

```
Print 1 ADD New_Queue
SPOOL 0 TO New_Queue
```

将队列连至打印机后，便可在“Print Queue Information”中“Currently Attached Servers”项窗口中查看当前的服务器名，若有名字，则当前的队列中的作业可打印，否则，作业不能被打印出来。

在批处理文件中可加入：

```
P  1  ADD  New_Queue  AT  PARIORITY  1
S  0  PRINTQ_1
```

表示 1 是打印机与队列名 New_Queue 连接，优先级为 1，缓冲区使用 0 号区。

参考文献

[1] 张怀莲编著,《IBM-PC 宏汇编语言程序设计》,电子工业出版社,1987 年。

[2] 沈美明等编著,《IBM-PC 汇编语言程序设计》,清华大学出版社,1991 年。

[3] 索梅等编著,《80386/80286 汇编语言程序设计》,清华大学出版社,1995 年。

[4] 刘乃琦编著,《IBM-PC 混合语言编程技术》,电子工业出版社,1990 年。

[5] 周明德编著,《微型计算机 IBM-PC(0520)系统原理及应用》,清华大学出版社,1994 年。

[6] 王士元编著,《汇编语言与外设编程》,南开大学出版社,1991 年。

[7] 刘甘娜等编著,《IBM-PC 微机原理及接口技术》,西安交通大学出版社,1993 年。

[8] 陈章龙等编著,《IBM-PC 机软硬件接口及实验》,人民邮电出版社,1993 年。

[9] 张载鸿编著,《微型机(PC 系列)接口控制教程》,清华大学出版社,1992 年。

[10] 刘乐善等编,《微型计算机接口技术及应用》,华中理工大学出版社,1993 年。

[11] 李继灿主编,《微型计算机原理及应用》,清华大学出版社,1994 年。

[12] 路友荣编著,《PC 系列微机接口扩展设计》,成都科技大学出版社,1994 年。

[13] 李伯成等编著,《IBM PC 微机应用系统设计》,西安电子科技大学出版社,1994 年。

[14] 钟玉琢等编著,《多媒体计算机技术》,清华大学出版社,1993 年。

[15] 何立民主编,《单片机应用文集》,北京航空航天大学出版社,1992 年。

[16] 钱涛宇等编著,《微机通信技术》,电子科技大学出版社,1992 年。

[17] 樊明传等编著,《微型机网络技术及应用》,四川大学出版社,1991 年。

[18] 王思华编著,《IBM PC/XT 多功能接口卡的设计》,计算机世界,1992 年第 6 期。

[19] 洪建荣等编著,《网络系统及应用》,西安交通大学出版社,1992 年。

[20] 朱家镗等编著,《微型计算机实用大全》,东北大学出版社,1993 年。